Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

1983

The national communications module of the stock point logistics integrated communications environment (SPLICE) local area networks

Carlsen, David D.; Krebill, Dan P.

http://hdl.handle.net/10945/19909

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

Downloaded from NPS Archive: Calhoun

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

Dudley Knox Library, NPS
Monterey, CA 93943

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THE NATIONAL COMMUNICATIONS MODULE OF THE STOCK POINT LOGISTICS INTEGRATED COMMUNICATIONS ENVIRONMENT (SPLICE) LOCAL AREA NETWORKS

by

David D. Carlsen
and
Dan P. Krebill

June 1983

Thesis Advisor: Norman F. Schneidewind

Approved for public release; distribution unlimited

T209078

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): The National Communications Module of the Stock Point Logistics Integrated Communications Environment (SPLICE) Local Area Networks	5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis, June, 1983
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): David D. Carlsen and Dan P. Krebill	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93940	12. REPORT DATE: June, 1983
	13. NUMBER OF PAGES: 158
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Defense Data Networks, Local Area Networks, Gateway, SPLICE, Internetworking, transport control protocol

20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This thesis discusses the development of an interconnection of Stock Point Logistics Integrated Communications Environment (SPLICE) local area networks with the Defense Data Network (DDN). The interconnection is done through what is called the National Communications (NC) module. After an introduction to SPLICE and DDN, a User's Manual for NC is presented, defining the environment and interfaces of NC. Then, the motivation behind the NC design and its operational context is explained in detail. Implementation issues are discussed next. Finally, the structure of NC is depicted by HIPO charts, pseudo-code and Ada language constructs.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Approved for public release; distribution unlimited.

The National Communications Module of the Stock Point Logistics Integrated Communications Environment (SPLICE) Local Area Networks

by

David D. Carlsen
Captain, United States Army
B.S., Brigham Young University, 1975

and

Dan P. Krebill
Captain, United States Army
B.S., United States Military Academy, 1973

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1983

Thesis
C268
c.2

y Knox Library, NPS
Monterey, CA 93943

ABSTRACT

This thesis discusses the development of an interconnection of Stock Point Logistics Integrated Communications Environment (SPLICE) local area networks with the Defense Data Network (DDN). The interconnection is done through what is called the National Communications (NC) module. After an introduction to SPLICE and DDN, a User's Manual for NC is presented, defining the environment and interfaces of NC. Then, the motivation behind the NC design and its operational context is explained in detail. Implementation issues are discussed next. Finally, the structure of NC is depicted by HIPO charts, pseudo-code and Ada language constructs.

TABLE OF CONTENTS

I. INTRODUCTION AND BACKGROUND OF SPLICE 8

A. DISCLAIMER 8

B. SPLICE BACKGROUND 8

C. SPLICE CONFIGURATION 9

D. SPLICE PROJECT AT NPS 12

E. FUNCTIONAL MODULE OVERVIEW 13

1. Terminal Management (TM) 13

2. Session Services (SS) 15

3. Database Management (DBM) 15

4. Recovery Management (RM) 15

5. Resource Allocator (RA) 16

6. Peripheral Management (PM) 16

7. Support Processes 16

F. THE DEFENSE DATA NETWORK 16

1. SPLICE Internetwork Alternatives 16

2. SPLICE Interconnection with the DDN . . . 18

3. DDN Protocols 19

G. THE NATIONAL COMMUNICATIONS MODULE 24

1. Overview 24

2. Design Goals of the National Communications Module 24

H. DESIGN APPROACH 28

II. NATIONAL COMMUNICATIONS MODULE USER'S MANUAL . . . 29

A. PURPOSE 29

B. OPERATIONAL CONTEXT 29

1. SPLICE Message Format 29

2. Server Processes 31

3. Functional Modules 32

C. INTERFACES TO NC 36
1. External Module Interfaces 36
2. Internal Module Interfaces 38
3. NC to NC Interfaces 40
D. OPERATION CONCEPTS 41
1. Message Flow 41
2. Users State Diagram 41
3. Error Conditions and Handling 43

III. NATIONAL COMMUNICATIONS MODULE DESIGN 45
A. INTRODUCTION AND DESIGN METHODOLOGY 45
B. LAN FUNCTIONAL REQUIREMENTS IMPOSED UPON NC . 46
1. Interactive vs. Deferred Message Traffic . 46
2. Server Processes 49
3. Session Services Module 53
4. Network Services Directory 54
5. Recovery Management 55
C. DDN-RELATED DESIGN ISSUES 55
1. Implications of Current SPLICE Design . . 55
D. DDN FUNCTIONAL REQUIREMENTS IMPOSED UPON NC . 57
1. Precedence and Security 57
2. TCP STATUS Command 58
3. Message Retransmission by NC 60
4. TCP Parameters: URGENT, Options 61
E. SUBMODULES OF THE NATIONAL COMMUNICATIONS MODULE 61
1. General Structure 61
2. Data Structures 62
3. General Functional Flow 63

IV. NC IMPLEMENTATION ISSUES 65
A. A MULTITASKING ENVIRONMENT 65
B. TCP/IP IN AN OUTBOARD COMPUTER 66
C. LANGUAGE ISSUES 67

D. INCREASING EFFICIENCY 68
E. FOLLOW-ON EFFORTS 68

APPENDIX A: HIPO CHARTS 70

APPENDIX B: NC ALGORITHMIC REPRESENTATION 123
A. NC_MAIN 125
B. OPEN 126
1. Interactive 126
2. Deferred 128
C. ESTABLISHED 129
1. Interactive 129
2. Deferred 130
D. CLOSE 132
1. Interactive 132
2. Deferred 133
E. ABORT 134
1. ABORT.OUT 134
2. ABORT.IN 134
F. ERROR_HANDLER 135
1. Error_Handler.OUT 135
2. Error_Handler.IN 135

APPENDIX C: NC DESIGN IN AN ADA SDL 136

APPENDIX D: SPLICE MESSAGE FORMATS 150

LIST OF REFERENCES 154

BIBLIOGRAPHY 156

INITIAL DISTRIBUTION LIST 157

LIST OF FIGURES

1.1 Proposed SPLICE Network System 10
1.2 Proposed SPLICE Software Configuration 11
1.3 Proposed SPLICE Local Area Network 14
1.4 DDN Host Interface Options 17
1.5 DDN Host Front End Processor Options 19
1.6 DDN Protocol Hierarchy 20
1.7 DDN Protocols and Corresponding ISO Protocols . 21
1.8 DDN Network Access Protocols 22
1.9 Physical Location of Interface Protocols 23
2.1 SPLICE Piggybacked Message Format 30
2.2 LAN - NC Message Flow 39
2.3 LAN - NC State Diagram 42
3.1 Functional Module and Server Process Hierarchy . 50
3.2 Information Returned by TCP STATUS Command . . . 59
3.3 National Communications Session Table 63
B.1 NC Module Detailed State Diagram 124
C.1 NC Task Graphic Representation 138
D.1 LAN Control Message 150
D.2 LAN Data Message 151
D.3 LAN Acknowledgement Message 152
D.4 LAN Piggybacked Data Message 153

I. INTRODUCTION AND BACKGROUND OF SPLICE

A. DISCLAIMER

The ideas and opinions expressed in this thesis are entirely those of the authors and do not necessarily represent the position of, or an endorsement by, the Naval Postgraduate School, Naval Supply Systems Command or Fleet Material Support Office.

Some terms used in this thesis are registered trademarks of commercial or government concerns. Rather than attempting to cite each individual occurrence of a trademark, all registered trademarks appearing in this thesis are listed below, following the firm holding the trademark.

Department of Defense, Washington, D.C.:

Ada

Digital Equipment Corporation, Maynard, Massechusetts:

VAX, UNIBUS, LSI-11, Q-Bus

B. SPLICE BACKGROUND

The present Navy Supply System consists of approximately 60 Stock Point (SP) and Inventory Control Point (ICP) facilities, all running the Uniform Automated Data Processing System - Stock Points (UADPS-SP) on Burroughs medium size computers (B-3500 through B-4800). Normally, each SP maintains two of these computers. The supply system also contains two ICPs, each using the UNIVAC U494. Many transactions entered into the supply system are processed in a batch mode. At several sites, some of the more specialized applications are processed on smaller "standalone" computers. Their output eventually is integrated with UADPS-SP using an offline entry system.

The Stock Point Logistics Integrated Communication Environment (SPLICE) is being developed to augment the Navy Supply Point and Inventory Control Point facilities. The main objective of SPLICE is to provide economical and responsive support capabilities by decentralized communications [Ref. 1,p.1-2]. To support this objective, SPLICE must meet two important needs. First, the increased need for interactive communication and second, the need to standardize supply site interfaces. Since input terminals play a major role in meeting both of these needs, SPLICE is designed to efficiently manage the myriads of input devices that are and will be available.

In short, the SPLICE subsystems will act as a front-end processor to the current Burroughs hosts, offloading communication processing from the hosts, and adding interactive processing.

C. SPLICE CONFIGURATION

The proposed SPLICE system will consist of standard hardware and software located in minicomputers at each SP and ICP. The minis will be tied together as a local area network (LAN) and each LAN will be connected together through the Defense Data Network (DDN).

The SPLICE project will progress in several phases, the final integrated system appearing somewhat like Figure 1.1 [Ref. 2,p.2-24]. Note that the Burroughs mainframes are supported by the SPLICE subsystems and are integrated into a LAN. The LAN at each stock point is tied together by a common internet--the DDN. There are also plans to add other hosts to some of the LAN's. [Ref. 2,p.2-23]

Figure 1.2 [Ref. 2,p.3-3] is the way the Fleet Material Support Office (FMSO) views the SPLICE software configuration at each LAN. The Burroughs hosts will be running in a

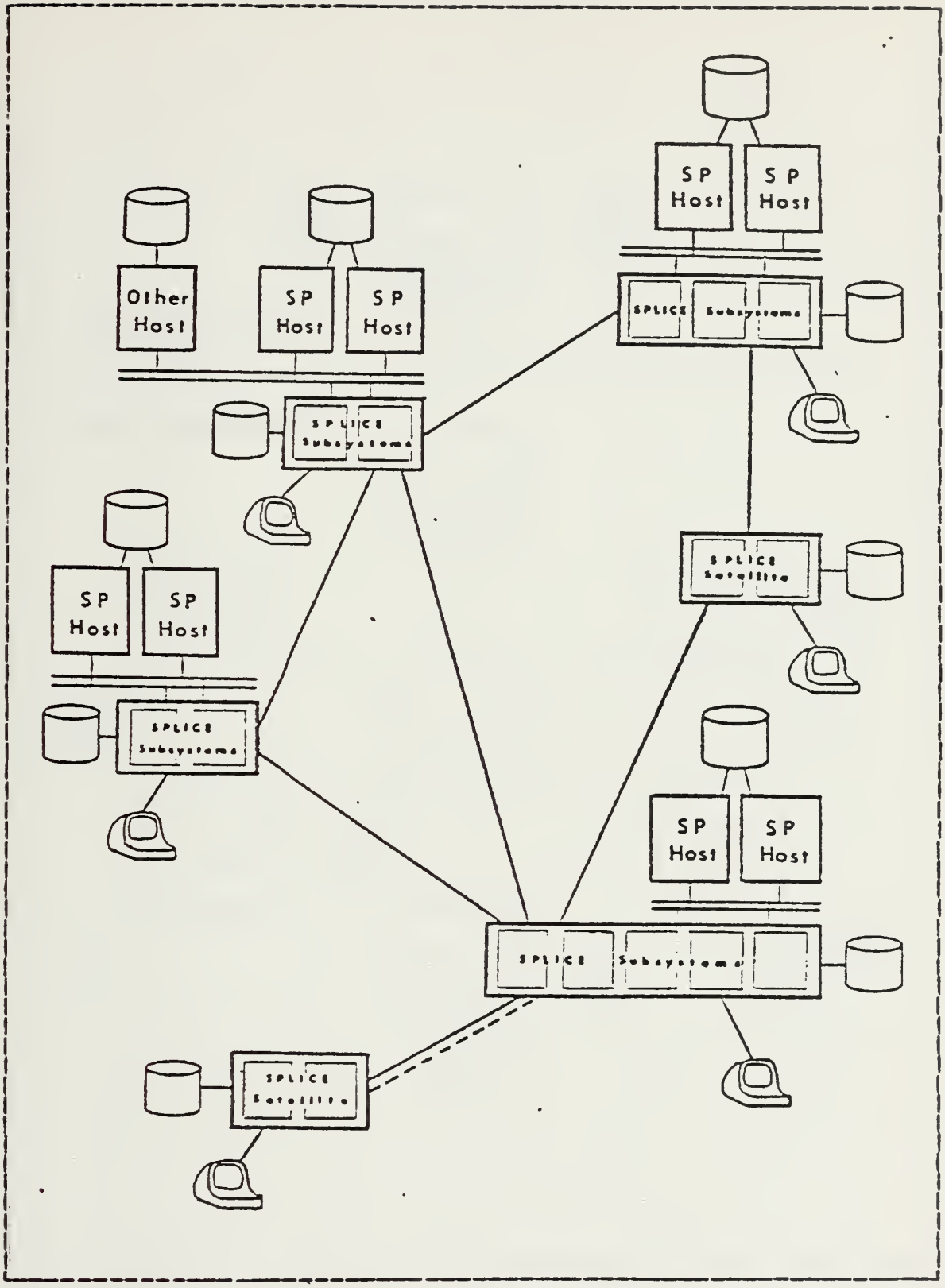


Figure 1.1 Proposed SPLICE Network System.

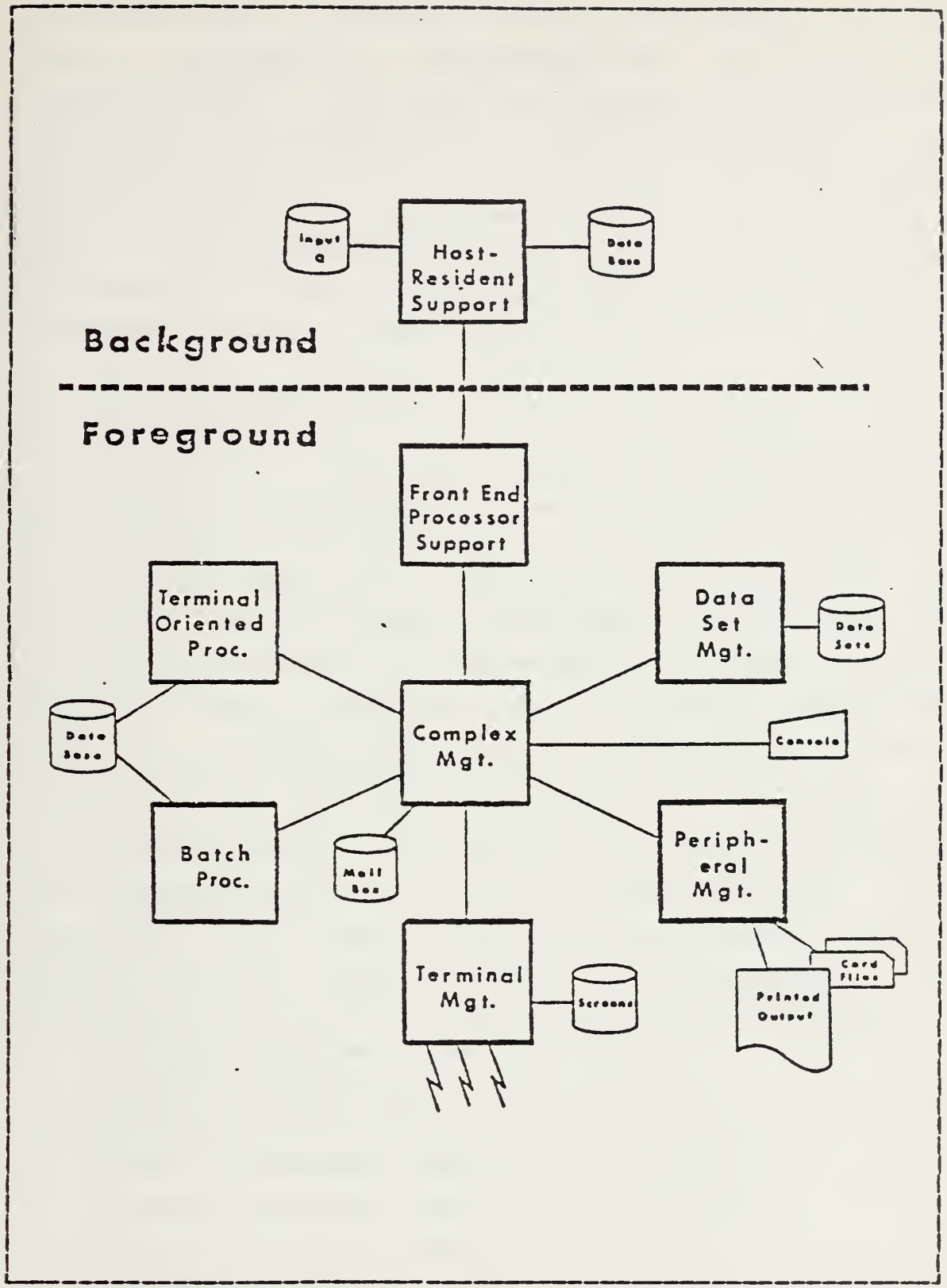


Figure 1.2 Propcsed SPLICE Software Configuration.

background mode, processing large file handling actions, while a complement of minicomputers will be running in a foreground mode. The minis will support the hosts with communications and interactive update capabilities. The envisioned software suite is centralized, with the Complex Manager acting as the arbitrator for all foreground activities. This implementation will provide for high system availability through a highly redundant hardware and software configuration. [Ref. 2,p.3-2,3-3]

D. SPLICE PROJECT AT NPS

In contrast to the FMSO design, the SPLICE functional design approach being used at the Naval Postgraduate School is directed toward developing a logical or virtual LAN first, thus ensuring that the functional requirements are satisfied. This is done by developing several functional modules, distributed in minicomputers throughout the LAN, with the necessary communications protocols to support them [Ref. 3]. All functional modules will operate independently, reacting only to messages from other modules. As opposed to the FMSC concept, there is no notion of a centralized manager. The NPS design allows for higher system availability than the centralized approach, since functional modules can be moved from one physical node to another, without changing their logical addresses. The proposed functional modules will include the following:

- Terminal Management (TM)
- Session Services (SS)
- National Communications (NC)
- Database Management (DBM)
- Recovery Management (RM)
- Resource Allocation (RA)
- Peripheral Management (PM)

Figure 1.3 shows the proposed physical view of the SPLICE LAN containing functional modules in a network of minis, communicating through a bus. Other supporting processes that are necessary for interface with SPLICE hardware include:

- Buffer Manager
- Local Communications
- Message Server

These processes will be contained in each of the physical processors in the LAN, corresponding somewhat to the "interface" portion of Figure 1.3. In addition, each functional module will have its own copy of the Message Server.

E. FUNCTIONAL MODULE OVERVIEW

The discussion in this section will be limited to a general overview of the SPLICE functional modules. Changes made to the module design in [Ref. 1], assumptions and interface considerations will be covered in a later chapters.

1. Terminal Management (TM)

This module includes terminal user message editing, screen management and virtual terminal operations. The Network Virtual Terminal protocol contained herein, will have the ability to convert differing terminal formats to a standard LAN format. It will also give the user the capability to design his own screen format, which will enable him to tailor his system to his own needs. Most of the network sessions will up between Terminal Management and some other module (most likely Database Management).

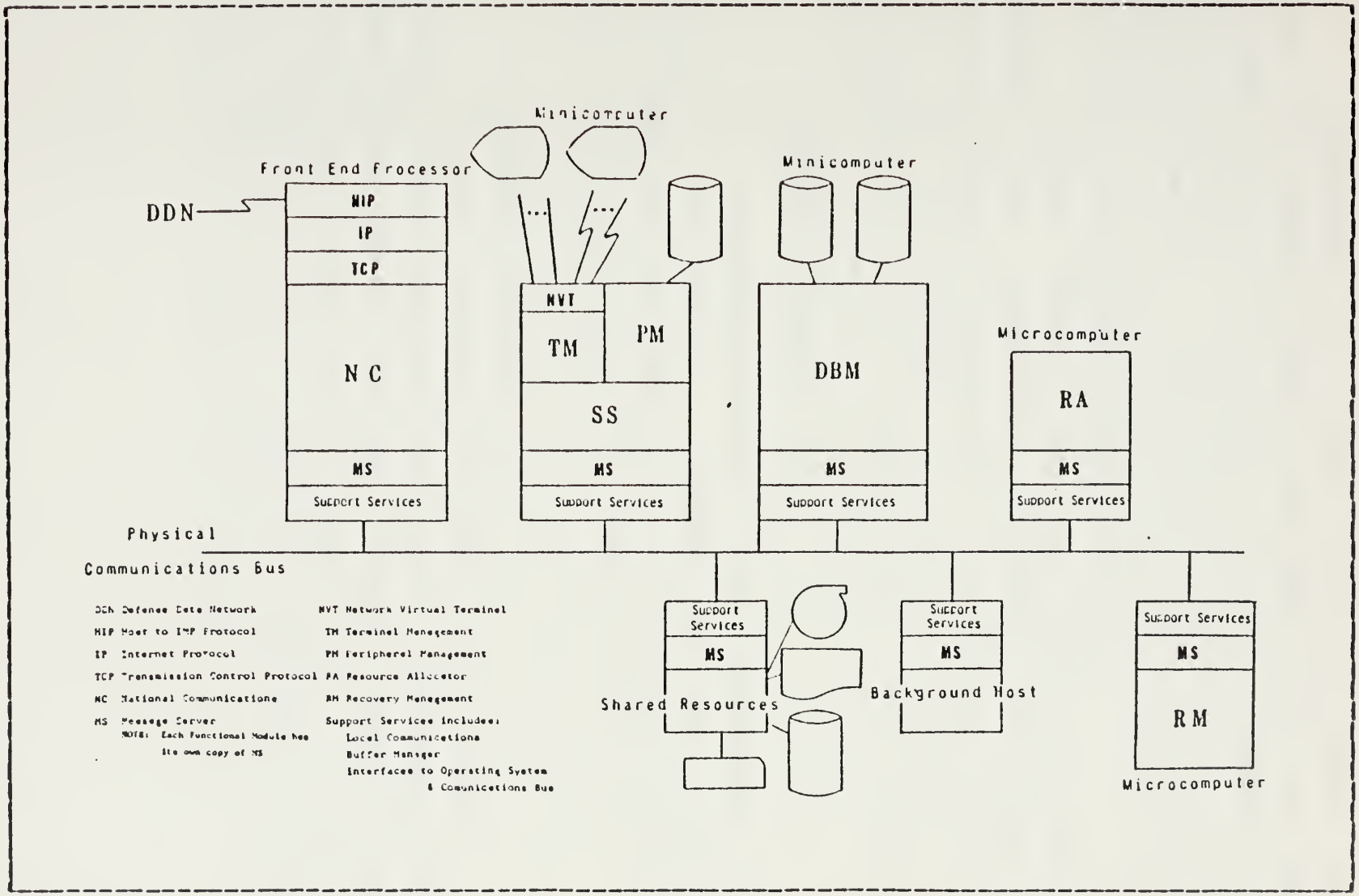


Figure 1.3 Proposed SPLICE Local Area Network.

2. Session Services (SS)

This module will be used to open and close interactive and deferred sessions with both local and remote modules. It will do this by retrieving both the logical and physical addresses of the destination module from a Network Services Directory. This directory, which will include the addresses of all modules within SPLICE, will be kept current by the Network Management Center for SPLICE.

Session Services will also contain the Mailbox Manager. This process will be invoked when incoming or outgoing messages are addressed to an inactive process or LAN. The Mailbox Manager will store the messages in the appropriate mailbox and deliver them to the destination process when it becomes active again.

3. Database Management (DBM)

The functions of the Database Management module consist of database query processing and file management. This module will be active in many of the SPLICE sessions, since SPLICE users will want to query or update their databases on a regular basis.

4. Recovery Management (RM)

The most important function of this module is to attempt recovery from error conditions. These errors conditions include reports from other processes concerning nonresponding SPLICE local modules or remote LANs. RM will also be responsible for Network Services Directory maintenance. This maintenance will normally be initiated by periodic updates from the SPLICE Network Management Center.

5. Resource Allocator (RA)

This functional module manages a pool of shared resources, including memory buffer space and disk drives. It will fill requests from individual Buffer Manager processes, allocating more main memory or disk storage space to a process that has exhausted its preallocated space.

6. Peripheral Management (PM)

The functions of this module will include processing commands to line printers and card readers and punches.

7. Support Processes

Local Communications, Buffer Manager and Message Server processes will exist as servers to the functional modules. They will act as the interface between the modules and the lower level hardware and software of the LAN. Their main functions center around handling message traffic.

F. THE DEFENSE DATA NETWORK

1. SPLICE Internetwork Alternatives

The interconnection of SPLICE LANs by some long haul means would result in a real improvement over the off-line, AUTODIN I communications which is presently used. Such a network might also allow for SPLICE users in one LAN to perform other new types of operations such as interactive, on-line queries of a database in a distant LAN. This all equates to the use of some long haul communications circuits with some means, as a minimum, to facilitate and manage the flow of data or messages between SPLICE LANs.

Options for providing these circuits include:

1. Establishing a dedicated point-to-point or a distributed network just for SPLICE use.

2. Contract to use a commercial network for specific services.

3. Utilize existing DOD systems.

Without going into details, suffice it to say that options 1 and 2 are less feasible than 3, especially considering factors such as cost, security and survivability. The SPLICE project accepted, near the outset, a commitment to use existing DOD systems. This selection may appear as a foregone conclusion but it should be emphasized that the other two options have been selected frequently by government

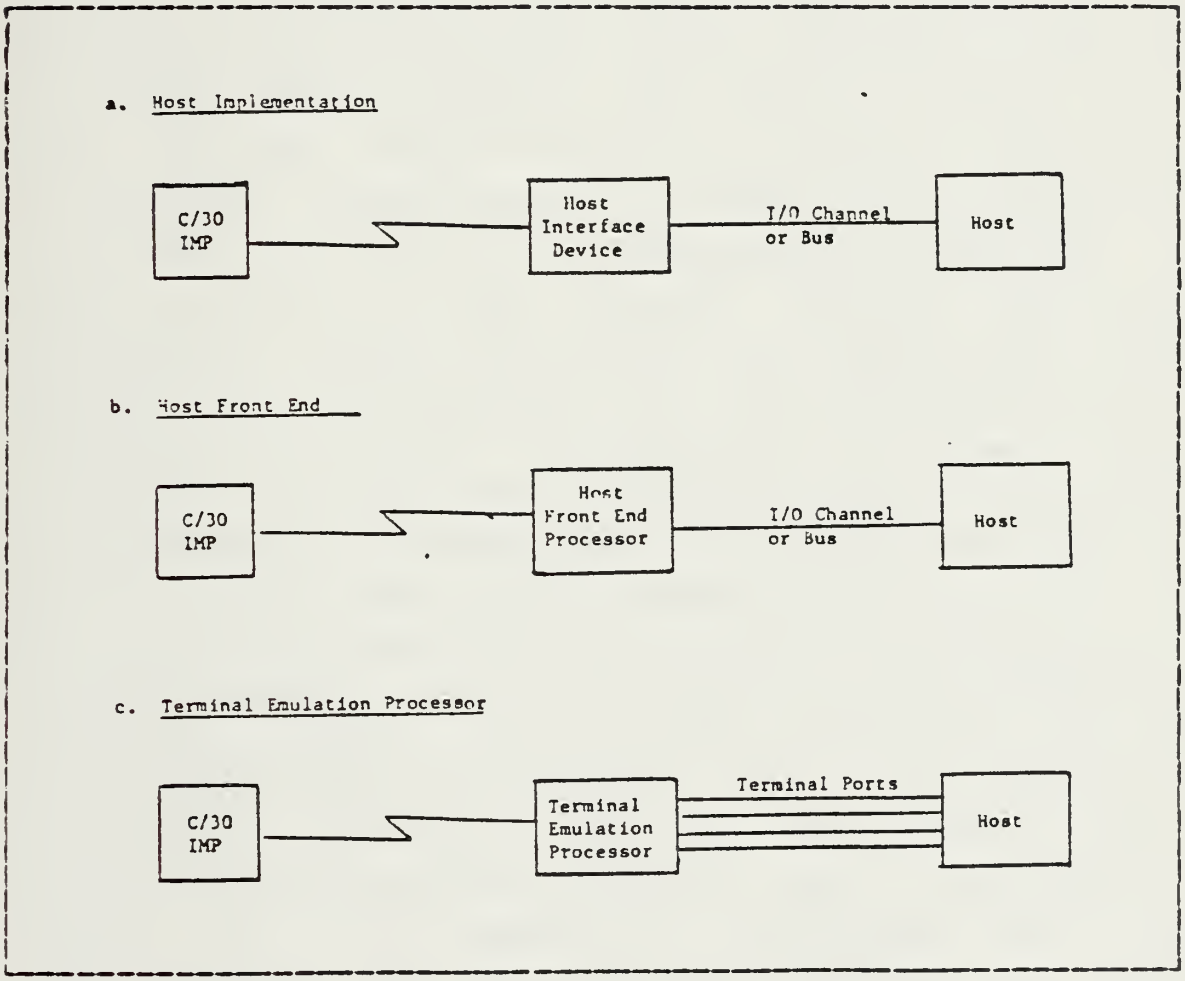


Figure 1.4 DDN Host Interface Options.

agencies faced with similar internetwork requirements. This trend was, in part, the motivation for the creation of the Defense Data Network (DDN). Previous SPLICE efforts focused on the DDN's predecessor, AUTODIN II [Ref. 2], but the current efforts are focused on utilizing existing ARPANET technology in developing the DDN.

2. SPLICE Interconnection with the DDN

As depicted in Figure 1.4, the DDN offers prospective users 3 basic interface options [Ref. 4,p.162]. The last option is not desirable as it greatly limits the DDN services accessable by the host. The direct interface contradicts one of the design goals of SPLICE: to off-load the communications overhead and software from the host. Thus, the most feasible option is to support a Host Front End Processor (HFEP) approach. The HFEP can be accomplished most easily by one of two standardized configurations as shown in Figure 1.5 [Ref. 4,p.56]. SPLICE's DDN traffic requirements are not anticipated to require high volume support. It is conceiveable that the interconnect will be supported by an LSI-11 microprocessor (or other) board installed in a SPLICE FEP to provide for part of the physical linkage to an IMP gateway. This option could provide for up to a 64 KB per second transfer rate.

One important distinction should be made concerning the HFEP approach. From the DDN point of view, the HFEP is supporting a "host". From the SPLICE perspective, the HFEP is the SPLICE Front End Processor (FEP), and in DDN terminology it is the "host." There is a requirement to implement the host-to-front-end protocols (HFP), shown later in Figure 1.9, in whatever is considered the host [Ref. 4,p.164]. For reasons stated earlier, the SPLICE approach is to off-load the SPLICE hosts, and to place them in the background [Ref. 1,p.1]. Consistent with this

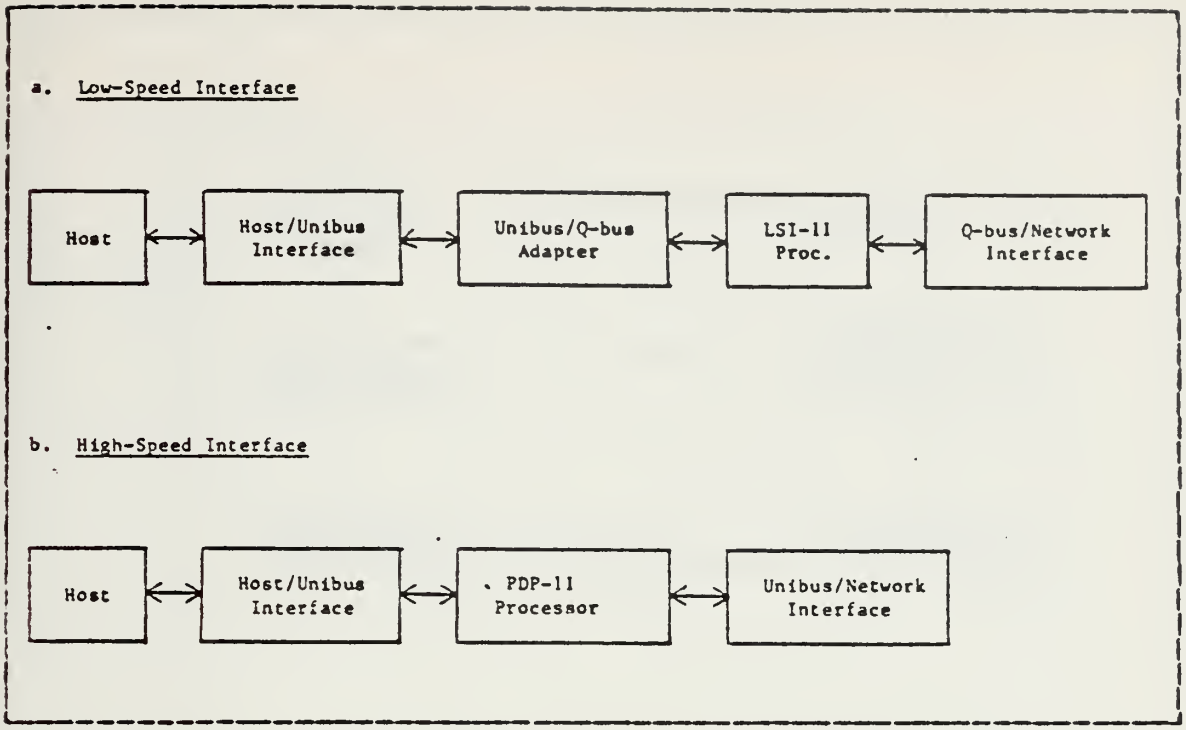


Figure 1.5 DDN Host Front End Processor Options.

approach is to consider the SPLICE FEP as the DDN host. One may even consider NC to be the software which supports the HFEP approach.

The actual interconnection of the HFEP would additionally require a high speed modem and a dedicated communications circuit to the nearest DDN IMP gateway. All of these physical interface components have considerable capacity for expansion.

3. DDN Protocols

a. Overview

Using the DDN naming conventions, four major groups of protocols exist. These are depicted in Figure 1.6 [Ref. 4,p.153] to represent the physical locations where these protocols reside. As contrasted with the ISO/CSI

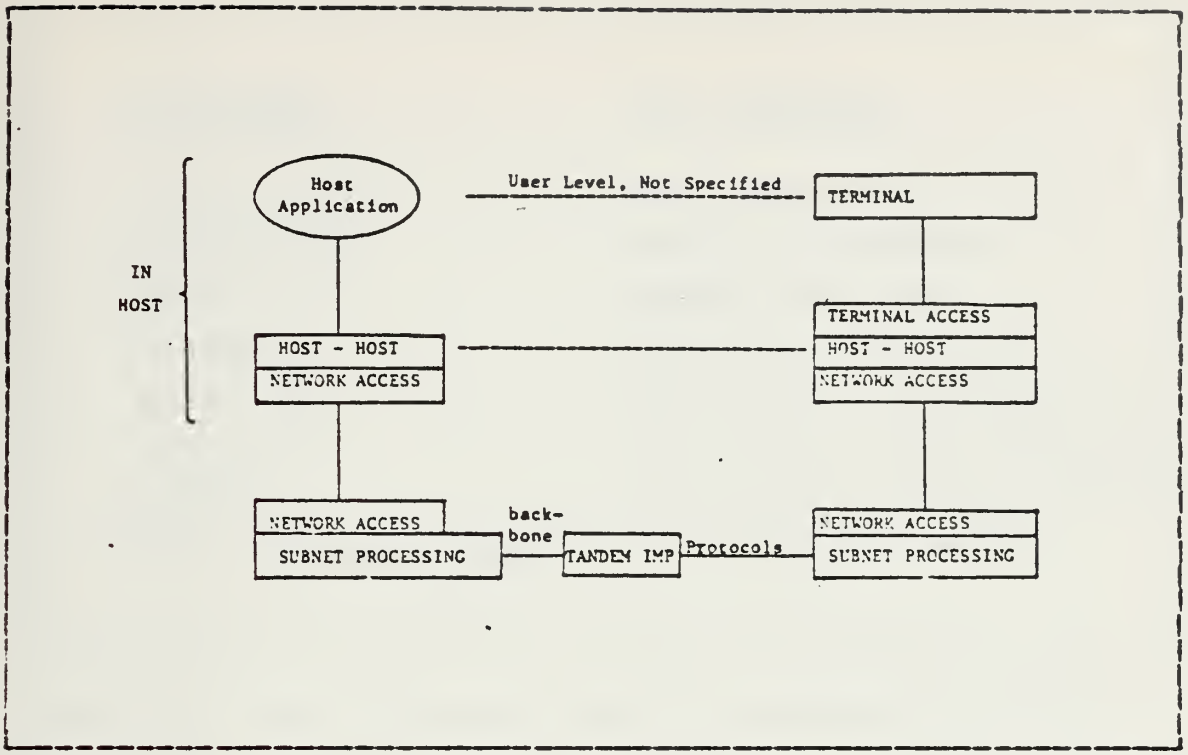


Figure 1.6 DDN Protocol Hierarchy.

reference model layers, these DDN names map to the model as shown in Figure 1.7 [Ref. 1,p.5-6].

b. Network Access Protocols

Network access protocols define the interface between a host and a switching node. Only hosts or their logical equivalent--such as a Front End Processor (FEP) of SPLICE--are supported directly by a switching node or IMP or gateway. These protocols encompass the physical, link and network levels of the OSI reference model. DDN provides a range of options for these layers of protocols, so as to accomodate the great spectrum of present and future users. These protocols are illustrated in Figure 1.8 [Ref. 4,p.156].

ISC Layers	DDN Protocols
Application	Same as for LAN
Presentation	Terminal Management
Session	Session Services
Transport	TCP/IP *
Network	DDN Options (SIP,X.25,etc)
Data Link	DDN Options (HDLC,1822,etc)
Physical	DDN Options (RS-232,422,etc)

* Note: There is no corresponding Internet Layer in the ISC Model.

Figure 1.7 DDN Protocols and Corresponding ISO Protocols.

Along the top of Figure 1.8 we see the general categories of protocols of the DDN design:

- The 1822 is a BBN developed protocol which is currently in use in the ARPANET C/30 IMPs and TACs.
- The HDH or HDLC Distant Host Protocol spawns from ISO's (HDLC) High Level Data Link Control, which is actually an offspring of IBM's Synchronous Data Link Control (SDLC).
- The Very Distant Host (VDH).
- The Segment Interface Protocol (SIP) is unique to the SACDIN interface. It uses the Advanced Data Communications Control Protocol which was derived by ANSI from IBM's SDLC.
- Supporting the X.25 standard should provide more latitude for SPLICE bidders and equipment vendors. This is not currently supported on the ARPANET and is a current design effort of the DDN.

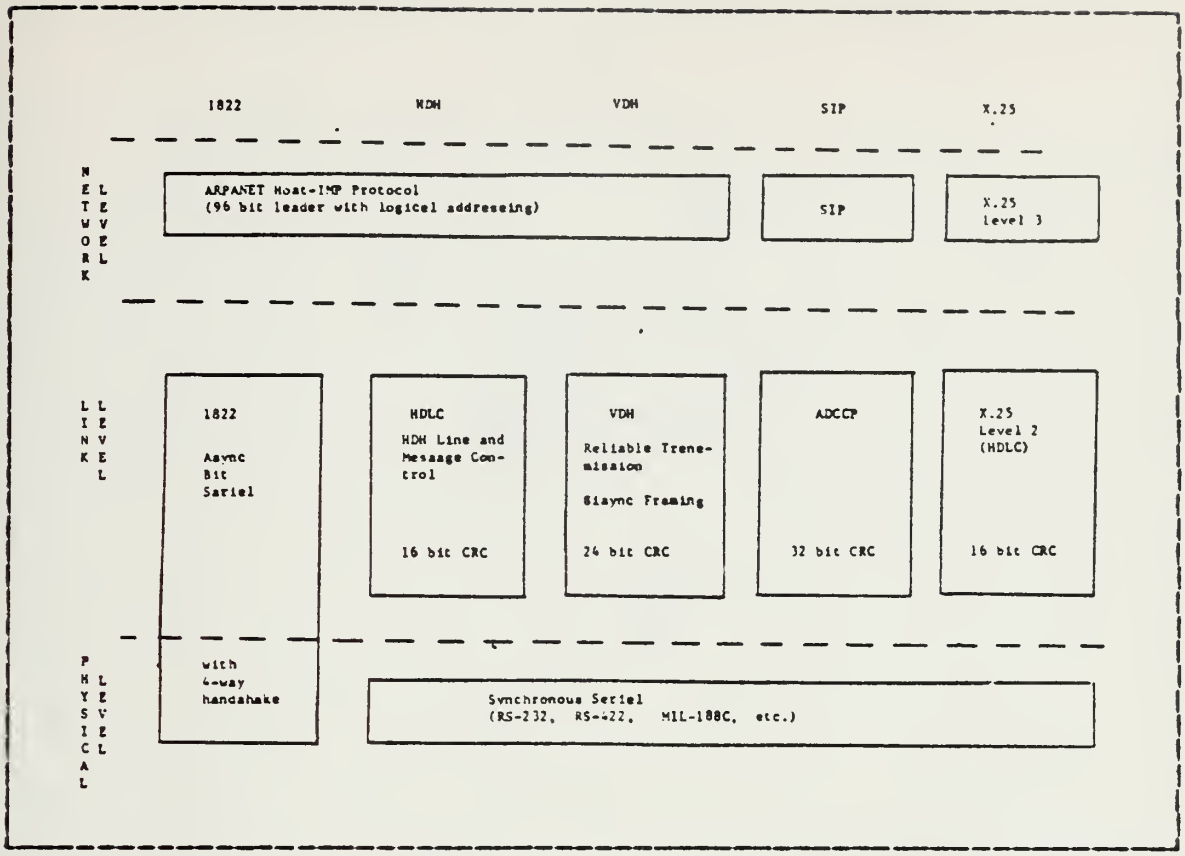


Figure 1.8 DDN Network Access Protocols.

All of these access protocols are interface protocols between the switching node and host (or SPLICE FEP) which it serves. They permit reliable transfer of data into and out of the subnetwork. They also provide varying levels of assurance that the data successfully traversed the network and provide subnet and subscriber status information. The protocols do not, however, provide for the reliable host-to-host or peer level communication. To support this and other aspects of the transport level protocols, DDN provides the Transport Control Protocol (TCP)/Internet Protocol (IP). These TCP/IP protocols could be implemented in the hosts themselves or, as in the expected SPLICE environment, within the FEP (or possibly, HFEP to the SPLICE

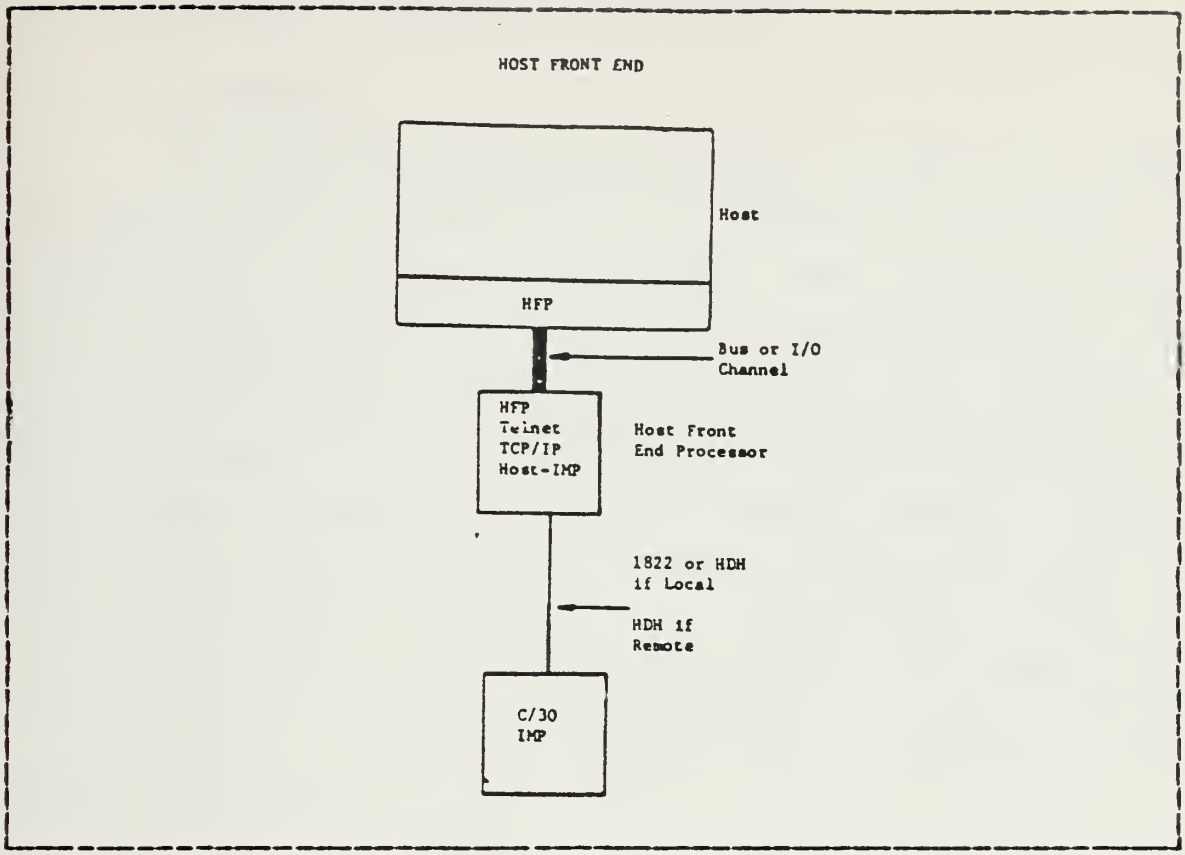


Figure 1.9 Physical Location of Interface Protocols.

FEP). One final way to depict these protocols is as shown in Figure 1.9 [Ref. 4,p.162]. Figure 1.9 represents the DDN concept and SPLICE will likely modify this in consideration of its intent to write its own application, presentation and session layers. The interface between SPLICE and the DDN will thus entail the correct interaction of the SPLICE National Communications Module and the TCP/IP protocols. The SPLICE FEP must provide the environment to support this interaction.

G. THE NATIONAL COMMUNICATIONS MODULE

1. Overview

The National Communications (NC) Module will serve to interface the SPLICE LAN to the DDN and vice versa. In this role as a "gateway" to the DDN, the NC also isolates the LAN from the DDN in that the DDN protocols will not have to be implemented internal to the LAN [Ref. 5]. Because of the inherent difference in speed of transmission between the LAN (1-2 MBPS) and the DDN (9600-56 KBPS), NC must control the frequent buffering of outgoing message traffic. Closely associated with the buffering control is the requirement for NC to provide for sequencing of LAN messages and message fragments. This is in part due to the requirement that there be only one unacknowledged message for a given session between any two NC modules in different LANs [Ref. 1,p.28,34].

2. Design Goals of the National Communications Module

Some underlying design objectives were developed for the NC module. These goals resulted from consideration of the SPLICE user needs as well as the planned operation of the DDN, primarily the Transport Control Protocol (TCP):

a. Transparency to User

The user should be provided with the illusion that there is only one global SPLICE network. The physical reality of multiple LANs tied together by the DDN should not be apparent to a user. Except for possible transmission delays, he will utilize a process in a distant LAN in exactly the same way he will utilize a local process. This will be accomplished by the "hidden" interaction of the user's process, Session Services, NC and the DDN. Thus, when a user on a terminal in a California-based LAN wants to

query an on-line database located in a Pennsylvania-based LAN, he will open his session in the same way as he would with his local database system. This requires the functioning of the NC and the DDN to be invisible to the user.

b. Interactive and Deferred Service

There may be times when a user does not desire interactive service with another process (local or distant). This is apparent when a user communicates with a background processor, where processing requests and transactions are queued for later batch processing. Another example similar to the previous database example, would involve a user who wants to submit a job to a distant database process. The job may be lengthy and he may want to do other things while the job is being processed by the distant database process. What the user should expect is that he can send a deferred (non-interactive) transaction to the destination database system. After the database system completes the transaction it should send back the results, which the user can obtain during a future (deferred) session. One of the basic properties of this service is that the user does not expect an interactive or quick turn-around response for his session message. This supports the notion that interactive sessions should be given preference over deferred sessions of equal precedence or priority. For NC, this means that interactive sessions should be given preferential handling over deferred sessions of equal precedence. This will help minimize the transmission delays that an interactive user might perceive if the NC is supporting his session.

c. Precedence

While one may surmise that most users of SPLICE will operate at a single level of precedence (probably Routine), higher priority capabilities should be provided.

The DDN will support Routine, Priority, Immediate, Flash and Flash Override. It is concievable that SPLICE requisitions under emergency or wartime conditions might require precedences up to and including Flash message handling. Therfore, the NC should be able to handle sessions of different precedences, and should clearly process messages of higher precedence before those of lower precedence. While this would need to be implemented throughout the LAN FMs, it has special connotations for NC.

One should note, while addressing issues of preferential handling, that there still is a need to handle control messages before any other messages [Ref. 1,p.26]. All this has the effect of establishing two sets of precedence queues and a special control message queue that all processes will maintain for servicing messages. The control queue will contain control messages and will be serviced before any other message, even a Flash message. The other queues belong either to those with interactive messages or those with deferred messages.

d. TCP Insurance

TCP is functionally specified to provide "...reliable interprocess communication between pairs of processes in host computers attached to distinct but interconnected computer communication networks" [Ref. 6,p.1]. TCP is intended to provide guaranteed sequenced delivery of message or message fragments over the DDN; however, SPLICE intends to provide a level of "insurance" on TCP's guarantee [Ref. 1,p.34]. Specifically, this will require the source NC to employ a stop and wait acknowledgement method whereby only one unacknowledged message or message fragment will be outstanding (with the destination NC) at any time. The source NC must await acknowledgement by the destination NC before any further messages will be sent for a particular

session. This equates to a high level NC to NC acknowledgement. This is done in addition to the acknowledgement and sequencing performed by TCP.

e. Insulate LAN from DDN Protocol Overhead

As mentioned previously, the NC "gateway" action also serves to insulate the LAN from the DDN protocol overhead. Very simple and, hence, very fast protocols can be utilized within the LAN and NC will support the transition from the simple to the complex protocols of the DDN. Therefore, our design should insure that little protocol overhead is induced on the LAN side in order to support the LAN's interconnection with the DDN. The barrier to this DDN overhead must be constructed within NC.

f. Design Independence and Modularity

This NC design is being attempted in the presence of some very key unknowns. The hardware and the operating system of the front end processor is not presently fixed nor is the software (or hardware for that matter) implementation of TCP known. One can speculate at great length about these unknowns and the design's critical efficiency could fail or succeed upon those unknowns. Thus, it is imperative that this design retain as much independence from the unknowns as possible. To that end, information-hiding techniques and data structure modularity are felt to be important design requirements for this stage of the project. Where possible, one should isolate the system-dependent calls or parameters to a single module. This will greatly ease any implementation of the design as well as improve its portability and maintainability. One must certainly note the arguments against such modularity:

> "...modularity is one of the chief villians in attempting to obtain good performance, so that the

> designer is faced with a delicate and inevitable tradeoff between good structure and good performance. Further, the single factor which most strongly determines how well this conflict can be resolved is not the protocol but the operating system." [Ref. 7]

This design effort cannot really begin to measure the effect of its modularity vis-a-vis the efficiency issues, and only an actual implementation will reveal such flaws. Nevertheless, the NC design is aimed at identifying and addressing all the NC requirements in a clear high level design structure. While the design may not ultimately reflect an efficiency-tuned implementation, it should provide a valid framework for the initial trial implementation.

H. DESIGN APPROACH

The National Communications (NC) module will provide for the use of the DDN backbone to interconnect dispersed SPLICE LANs. This thesis examines how NC will accomplish this by looking first at the interface on the LAN side of NC. This amounts to a type of "user's manual" where users are the other Functional Modules within the LAN. The requirements of the NC module and its subcomponents are examined next. Finally, some of the pertinent implementation issues concerning the NC and other SPLICE modules are addressed. This work is intended to explore the details of the SPLICE modules (especially NC), and as a first effort towards actual implementation, one must anticipate that the NC design will evolve with later work on the project. A number of previous SPLICE concepts needed careful compromising and refinement as the NC design evolved. It is thus expected that future work will result in similar design tradeoffs as the other SPLICE modules enter a more detailed design stage.

II. NATIONAL COMMUNICATIONS MODULE USER'S MANUAL

A. PURPOSE

This chapter will give the user of the National Communications (NC) module a general overview of the context in which NC operates and the information necessary to interface with the NC module. It is important to note that a user of NC is considered to be any person or process that must communicate with NC.

The intent of this user's manual is to describe how NC and its users interact, but not why they operate as they do. Reasons for, and alternatives to, the mode of operation discussed in this chapter will be covered later in chapter 3.

B. OPERATIONAL CONTEXT

It is felt that in order to gain an appreciation of the interfaces to NC, one should first understand the overall context in which NC operates. This section is designed to give a more detailed look at the operations of the SPLICE functional modules. It will also serve to highlight the changes that have been made to the design of the SPLICE LAN since [Ref. 1] was published.

1. SPLICE Message Format

The SPLICE LAN will operate on the concept of sessions set up between functional modules. Each session will be assigned a unique session number with that number being used in all messages and acknowledgements during that session. Precedence and classification information are also included in the each message. Message acknowledgements can

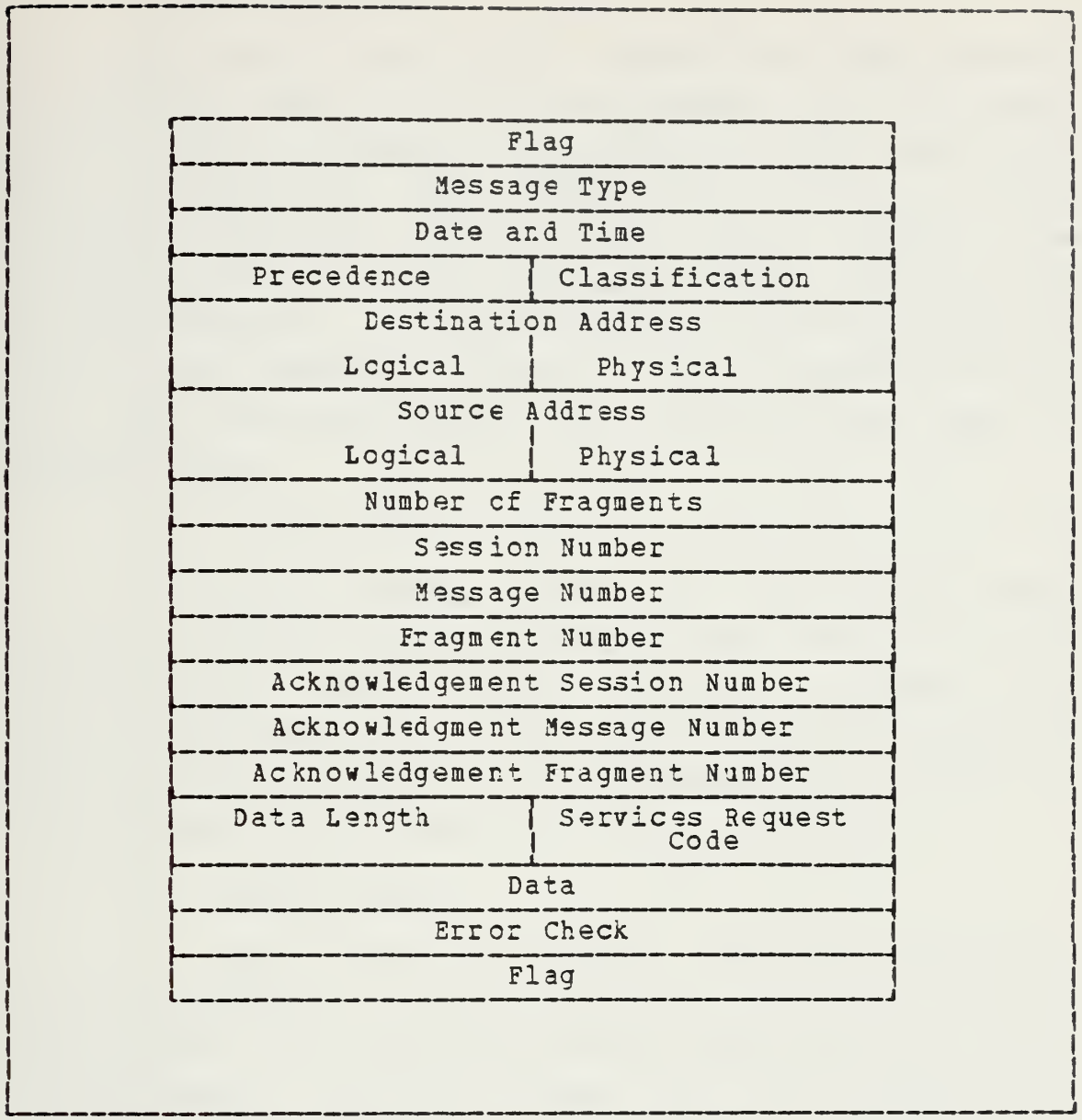


Figure 2.1 SPLICE Piggybacked Message Format.

either be sent separately or piggybacked on any other message going to the same destination module. The message format for a SPLICE LAN piggybacked message is shown in Figure 2.1.

2. Server Processes

As the LAN cperational context was being defined, some general functicns that are necessary when any functional mcdule processes a LAN message were identified. These functions have been grouped into the three server processes described below. The intent behind these processes is that at least one copy of each exists in each physical processor, and their functions are concurrent with those of the functicnal modules. It is important to note that each functional module will have its own copy of Message Server.

a. Local Communications

Local Ccmmunications (LC) will most likely consist of vendor supplied protocols for the lower two or three levels in the ISO model. It will be present in all physical processors as the interface to the communications bus. LC will be invcked by the Message Server on behalf of the functional module desiring to send a message.

b. Buffer Manager

This process will handle allocation and deallocation of buffers, keeping a table of active and inactive memory space. If a functional module runs out of its preallocated memory, the Buffer Manager (BM) will automatically request shared storage from the Resource Allocator. The Message Server will be the the process which uses BM the most. NC will also use it for deallocation of DDN message buffers.

c. Message Server

The Message Server (MS) is a group of related message handling functions that will be colocated with each

functional module. The functional module will initiate a local send with a pointer to the buffer containing the message to be sent. MS then will take over, adding information provided to it by Session Services at the beginning of the current session. This information consists of a session number, a precedence, a classification, and the source and destination addresses. MS will also add the message number and then put the message on its applicable precedence queue. Interactive message queues will be separate from deferred message queues, with interactive queues being served before deferred queues of the same precedence. When the message comes to the front of the queue being serviced, the message will be given to LC to be sent out on the LAN communications bus. MS will handle assembly and disassembly of multi-fragment messages, maintaining control of an outgoing message until the last fragment has been acknowledged. When the message acknowledgement arrives from the destination process, MS will deallocate the message buffer. During the period between a message and its acknowledgement, no new messages may be sent in that particular session. If the message remains unacknowledged after an appropriate timeout period, MS will retransmit the message. If no acknowledgement is received after retransmitting the message twice, MS will inform Recovery Management and Session Services of the nonresponding functional module. Session Services will prepare an error message for Terminal Management to present to the user and the session will be aborted.

3. Functional Modules

This section provides a general overview of the SPLICE functional modules and how they interact. It is not meant as a comprehensive view of all the modules' functions; therefore, many of the details are left for future theses. In each functional module section is a general description

of how it handles a message and sets up and maintains a session.

a. Session Services

A SPLICE session can be defined as a network activity which involves two or more functional modules, all reacting to the same message or set of messages. A session is initiated by an open message to Session Services (SS) from one of the functional modules. In most cases the sessions will be initiated by Terminal Management, but there may be occasions when another module (e.g., Database Management) is allowed to initiate the session. During the process of setting up the session, SS will look up the logical and physical addresses of the destination process and determine if it is reachable. If it is not reachable SS will handle the session through a mailbox, where it will store the user's messages and try to send them later. If the destination is reachable, SS will add a new session to its session tree, assigning a unique session number to it. Precedence and classification of the session will be determined by source process specification, or by network default. SS will send the addresses, the session number, the precedence, and the classification to the source and destination processes, where they will be stored by the respective Message Servers. Sessions with remote processes outside of the LAN will be handled through NC, and will be discussed in more detail later. When the session is set up, SS has no further interaction with the communicating processes until the session is aborted or closed. Each communicating process sends its messages directly to the destination, with the source Message Server appending the proper information to the messages. A close or abort message will cause the session to be terminated and deleted from the session tree. Only those processes involved in a

session, (with the possible exception of Recovery Management), should be allowed to abort that session.

b. Terminal Management

Terminal Management (TM) is the terminal user's interface with the LAN. As explained previously, the Network Virtual Terminal protocol is contained here, enabling the user to communicate with other users on the LAN regardless of the type of terminal they each have.

When the user first attempts to log on, TM will verify the user's password, and if it is correct, the user will be allowed to create a session with the functional modules of the SPLICE Network. When opening a session, the user will be asked to give the classification and precedence levels that are desired for that session. Once the classification and precedence levels are set, they will remain fixed throughout the entire session. If the user is authorized to operate at the levels he requests, he will be allowed to begin sending session messages. First, TM will convert the user's messages to Network Virtual Terminal format and then notify Session Services of the user's intentions. If the user wants an interactive session, a full duplex connection will be set up with the destination functional module. If the user wants a deferred session, TM will deposit the message into the appropriate mailbox and request Session Services set up a session between the mailbox and the destination functional module.

c. Recovery Management

This module will handle LAN error conditions, to include all the error messages from functional modules reporting a nonresponding destination module. Recovery Management (RM) will check the status of the questionable module, and inform the Network Management Center (NMC) if

the module is no longer reachable. RM will also be receiving periodic update messages from the NMC concerning the status of all the other LAN's within the network. These changes will be entered into the Network Services Directory, to which RM will be the only module with write access. RM will have built into it a statistics gathering process that will either actively or passively obtain LAN and DDN information that might be required by the NMC. Opel's thesis [Ref. 8] contains a detailed discussion on the possible implementations of the statistical portion of RM. RM will also be responsible for routine or emergency backup of the LAN state for recovery purposes, and a graceful shutdown of the LAN, if necessary.

d. Resource Allocator

The Resource Allocator's (RA) main function will be to manage a pool of extra memory and disk space to be used when a process exceeds its normal message buffer and workspace allocation. RA will be called by Buffer Managers throughout the LAN to obtain and dispose of extra resources.

e. Other Modules

An explanation of the internal operations of the other functional modules, which include Database Management and Peripheral Management, will be deferred to later theses. Their general functions are defined in [Ref. 1]. The important point to note here is that these functional modules will be sending messages through NC, and the actions of their respective message servers will be the same as has been described above.

C. INTERFACES TO NC

Interfaces to NC can be viewed from three broad contexts:

1. Interfaces for user modules physically external from the processor supporting NC but local to the LAN. These interfaces entail control or data messages sent or received via the LAN communications bus.

2. Interfaces for modules physically internal to the processor supporting NC. These interfaces involve parameters associated with either control or data messages that are passed via the operating system or run time environment of the processor.

3. Interfaces between NC modules in different LANs. These interfaces are accomplished using data messages that are sent or received over the virtual circuit provided by the DDN.

These interfaces form a group of user's protocols with which modules may access and utilize the services of NC (as well as other SPLICE modules).

1. External Module Interfaces

The SPLICE modules that interface with NC include those which need to send or receive messages to or from distant SPLICE LAN Functional Modules (FM). Terminal Management (TM), Session Services (SS), Recovery Management (RM), Database Management (DBM), Peripheral Management (PM) and Resource Allocator (RA) can all initiate sessions (interactive or deferred) that require the interface with NC. One must consider that NC may also need to interface with any of these FM's mailboxes as well. The support modules known as Message Server (MS) and Local Communications (LC) were examined earlier. These modules play key roles in the higher level interface between NC and its local FMs.

As discussed earlier, when an FM wants to utilize NC, it does so through its MS and its LC modules supporting the particular processor in which the FM resides. Similarly, NC communicates with other local FMs through its copy of MS and LC. What is passed through each respective layer of MS and LC are predefined message formats (see Appendix D). MS will attempt to send the message to its correct destination and, in so doing, it will handle precedence and acknowledgements on behalf of the FM it supports. As a rule of thumb, local acknowledgements are performed at the MS to MS level. The basic categories of these messages provide a framework for discussion of the FM interfaces, and are as follows:

a. LAN Control Messages

These messages will be routed via the logical control bus [Ref. 1,p.26], and will contain a response string that will be interpreted by the receiving FM (including NC). Control messages are utilized for the more critical control actions such as system recovery, shutdown, error conditions, etc., that demand immediate action by the FM. For example, one of the control messages that NC will send or receive is a session ABORT message. The NC design requires that control messages will be processed before any data messages.

b. LAN Data Messages

These messages will be routed via the data bus and contain a data portion of interest to the receiver. This format should also be used for most message/message fragment traffic where the sender has no acknowlegements to include or "piggyback" with the message. This format will also be used by NC and the respective FM for setting up (opening) and terminating (closing) sessions. Additionally, NC will use this format for reporting errors to users and Recovery Management.

c. LAN Acknowledgement Messages

This message is routed via the data bus and contains no data. It is used by the receiver of a data message when there are no other data messages addressed to the source functional module, on which to piggyback the acknowledgement. This format may frequently be used for Deferred sessions.

d. LAN Piggybacked Messages

This message is routed via the data bus and combines the data and acknowledgement information described above. It is utilized at the Message Server level whenever possible. Note that the acknowlegement must also contain a session number (see Appendix D).

2. Internal Module Interfaces

The NC module may be visualized as the controller between the DDN (TCP module) and the LAN (MS module), as shown in Figure 2.2. MS functions as NC's access to the LAN and TCP functions as NC's access to the DDN.

The MS interface to NC is virtually identical to the interface between MS and the other FMs elsewhere in the LAN. Parameters such as buffer pointers to messages are passed between NC and MS. Conceptually, MS is a process executing concurrently with NC (and TCP), and communication (parameter passing) between NC and MS is accomplished through the operating system of the processor. MS, NC and TCP are perceived as sharing a common memory managed by the Buffer Manager. Besides performing allocation and deallocation of buffers for the FMs, Buffer Manager should also support deadlock detection capabilities. MS is assumed to handle precedence and control messages in the same fashion as described earlier for NC, with data message precedence queues and a separate control message queue. As contrasted with the NC to

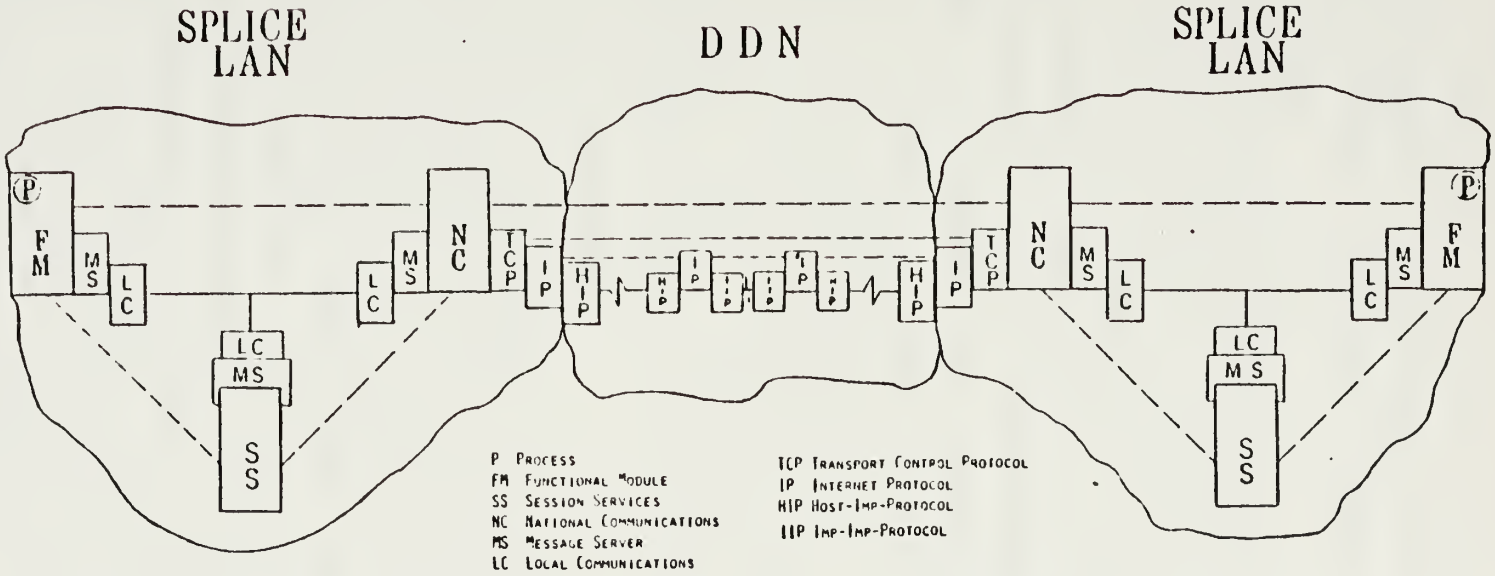


Figure 2.2 LAN - NC Message Flow.

TCP interface, messages passed between NC and MS are not handled by a stop and wait acknowledgement method.

No acknowledgements will be performed between NC and MS. This was felt to be practical because NC and MS will likely be implemented in the same physical processor, and communications between the FMs is supported via operating system facilities.

Communications between NC and TCP will be accomplished as for MS--through the operating system. The TCP to NC interface must rest upon the minimum TCP functional specification as contained in [Ref. 6]. More details of these predefined interfaces are contained in subsequent chapters and the appendixes. SPLICE requirements, as addressed earlier, added the stop-and-wait acknowledgement method. The dichotomy of interactive versus deferred sessions is also supported but this distinction is not present in TCP; rather, the distinction is enforced by the way that TCP is utilized by NC. Five TCP commands are required: OPEN, SEND, RECEIVE, CLOSE and ABORT. The optional TCP STATUS command was not incorporated as part of the NC design.

3. NC to NC Interfaces

The NC to NC interface is accomplished over the virtual circuit provided between the source and destination TCP modules. This interface is utilized at various stages of sessions, and serves to synchronize the NC modules so as to reliably control message flow between LANs. The most basic part of this interface involves the stop and wait acknowledgement scheme discussed previously. The source NC will control sequencing of messages to the destination NC by always waiting for a destination NC acknowlegement on every data message. While this detracts from the throughput of the NC module, it provides another layer of insurance upon the TCP guarantee of sequenced delivery. As detailed in the

next chapter, this acknowledgement scheme does not include retransmission by the source NC.

D. OPERATION CONCEPTS

1. Message Flow

To provide a graphic illustration of the message flow, Figure 2.2 depicts the general relationship between modules previously described. LAN messages to and from Session Services (SS) are required as part of opening sessions in each LAN. After the sessions are established, the actual data message is routed from the functional module (FM), through the respective Message Server (MS) and Local Communications (LC), and then the source National Communications (NC) module interjects the message through TCP to the DDN. Something of a reverse sequence takes place at the distant LAN before the message finally arrives at the destination FM.

2. Users State Diagram

Another graphic means of depicting the operation of NC is shown in Figure 2.3. Figure 2.3 was intended to describe the general states of NC, but it also captures the general states applicable to intra-LAN message traffic not involving NC. The LISTEN state is the state where NC has initialized TCP to be ready for any incoming or outgoing messages. NC is likewise ready to react to service any outgoing message requirement received from its MS. In general, a particular FM must first initiate actions to OPEN a session (interactive or deferred) with a destination FM before any actual data messages may be sent. This requires an interchange of information between the source FM and SS. After insuring that the destination FM is reachable, SS provides the destination's physical address to the source

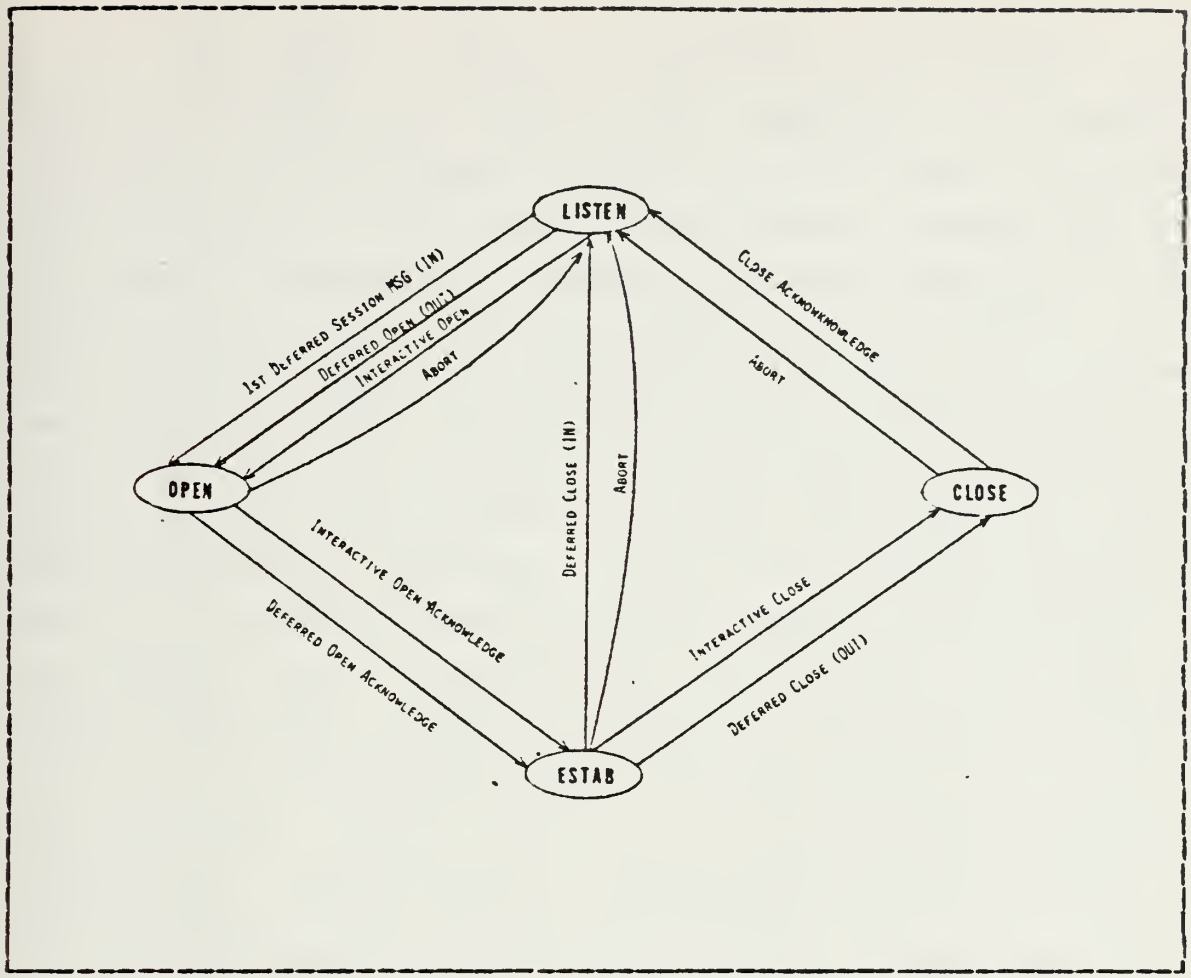


Figure 2.3 LAN - NC State Diagram.

FM's Message Server. After the destination FM acknowledges a new session requirement, the state changes to ESTABLISHED during which the data message flow is accomplished.

The exception to this sequence of steps is for an incoming (from the DDN) deferred session requirement. For this, the source NC will actually be sending the first data message to initiate a deferred OPEN in the destination LAN. The destination (receiving) NC then sets up a deferred session with the destination FM's mailbox and finally acknowledges the source NC. This changes the state to ESTABLISHED.

One should note the two actions which change the state from ESTAB to LISTEN. In one case, NC has just received a "Deferred Close (IN)" message on a deferred session. If this message is found to be in proper sequence, the receiving NC knows that it has already received all of the stream of messages or message fragments associated with a deferred session. No more messages will follow. The destination NC can thus acknowledge the source NC and then return to the LISTEN state. There is no need for the destination NC to cycle through the CLOSE state. In the other case, NC has just received an ABORT, and like all other ABORT processing from other states, NC takes expeditious action to terminate activity on that session, and returns to the LISTEN state. Analogous to TCP actions, NC does not acknowledge ABORT messages. ABORTs may be called as a result of source or destination user processes, TCP or DDN interaction or other causes such as LAN shutdown or recovery. Under these circumstances, acknowledgement of an ABORT is not necessary or may even be undesirable.

The other state changes are more intuitive, based upon either user CLOSE or ABORT messages. This state diagram is not meant to reveal the finer details of how NC accomplishes these changes in state. Subsequent chapters will explore these details, and Appendix B examines the details in full.

3. Error Conditions and Handling

To exhaustively catalog the various probable errors is beyond the scope of this thesis and the level of design at this stage of the project; however, a general description of errors and error handling is provided for NC users.

Most errors fall into one of two general categories--fatal and non-fatal.

a. Fatal Errors

These errors characterize a state where no further useful processing can be done. One should put this notion in the context of each individual user session instead of a fatal error to the entire processor. For the former, specific actions may be taken upon recognizing the error, and the session will be ABORTed. For the latter, one may visualize that Recovery Mangement would have deadlock detection facilities and would initiate actions to reboot the erroneous processor. Locally-generated errors that are not detected or corrected by LC or MS will be fatal errors (if detected) to NC. NC is also not very tolerant of errors detected from the TCP interface. The fundamental action for such errors is to inform the user of the error, report the error to Recovery Management and ABORT the session.

b. Non-Fatal Errors

These errors are ones from which recovery is very likely. Such is the case when LC does not get an initial acknowledgement on a message sent, MS receives a duplicate copy of a message, or Buffer Manager runs out of dedicated buffer space. The processing situation has degraded, but may be continued. For NC, almost all locally-generated errors should be detected by its LC or MS modules. Furthermore, NC has virtually no capability to deal with such errors except to treat them as fatal and sequence through the ABORT outlined previously; however, on the other side of the NC gateway, NC does have a limited capability to deal with one TCP error: "Insufficient Resources" [Ref. 6,pp.54-58]. For this condition, NC can attempt, at least two more times, to re-invoke TCP. If those attempts are unsuccessful, the error becomes fatal and NC initiates an ABORT for the session.

III. NATIONAL COMMUNICATIONS MODULE DESIGN

A. INTRODUCTION AND DESIGN METHODOLOGY

This chapter will examine the major design decisions that were made while deriving the National Communications (NC) module. Several design decisions were first required to build a detailed concept of operation of the LAN, and these decisions formed a basis for determining how NC should be constructed to interface with the LAN. Similarly, a concise determination of how TCP functioned was needed before NC could be designed to access and be accessed by the DDN. Indeed, the NC design is largely shaped by the nature of its inputs and outputs because these must dovetail with both SPLICE and DDN design specifications.

The design methodology employed for this task was based on a structured analysis approach. The primary design aid utilized for this methodology was HIPO, which stands for Hierarchy, plus Input, Process, Output [Ref. 9]. HIPO had previously been used for the SPLICE project, [Ref. 10] and [Ref. 11], and provides a flexible medium for expressing sequential designs for input and output oriented systems. Detailed HIPO diagrams are provided in Appendix A. The underlying basis of the HIPO technique was functional decomposition in a top-down approach. Also, some Information Hiding design concepts, [Ref. 12] and [Ref. 13], were used, as appropriate. These concepts are particularly desirable when one considers the need for SPLICE modules to be easy to modify. Changes to SPLICE operations such as message format modification are likely to occur over time. Effective modularization in the design phase will minimize the impact later on for such changes. This was one of the reasons, for

example, a new server process known as Message Server was abstracted cut of the design of the LAN.

One of the other products of the NC design effort was the develcpment cf pseudo code (Appendix B). Additionally, an attempt was made to place the sequential HIPO design into a ccncurrent and multi-tasking environment by using the programming language Ada (Appendix C) as a Systems Design Language (SDL). HIPC charts, pseudo code and the Ada SDL were the end products of the design. The rest of this chapter will highlight the key design decisions that were made in the process of developing those products.

B. LAN FUNCTIONAL REQUIREMENTS IMPOSED UPON NC

1. Interactive vs. Deferred Message Traffic

One of the basic assumptions made about SPLICE is that there will be both interactive and non-interactive (deferred) message traffic present in the network (see chapter 1). It is logical to assume that users will sometimes want to get a response right away, whereas, other times, they will have neither the time nor the inclination to wait for the answer.

An example of an interactive session is one in which a terminal user calls up the Database Management (DBM) module tc find out the current status of a stock item. The message is sent to DBM, which responds quickly by sending a message containing the information requested to Terminal Management (TM). TM then presents it to the user in the proper screen format.

Examples of deferred sessions include those in which a terminal user has a large number of transactions to be processed in a batch mode by one of the functional modules or by the background host processor. He might also want to send some "mail" to another terminal user, requesting his

assistance on a problem. In these deferred examples, the users do not need instant response--an answer in several hours is probably sufficient. In the case of database updates, a response may not be required at all.

One way to handle interactive and deferred traffic is to require no distinction between them. Each message of a certain precedence is put onto the a single queue of that precedence, regardless of its session type (deferred or interactive). Consider, however, the case when a user releases a large routine deferred message of several hundred transactions just prior to another user attempting to send a short routine interactive message. The response time for the interactive user would degrade to a point where he might wait several minutes before he gets any response at all. Opening a session is even more critical, especially when the traffic volume is heavy and there are few resources left for creating local connections on the DDN. An interactive session clearly should get priority over any deferred session.

a. Interactive First, Deferred Second

With both interactive and deferred traffic competing for resources, it becomes necessary that the two be kept separate. Interactive messages should get quick response, whereas deferred traffic can wait.

Two sets of message queues have been developed within NC and the Message Servers. These queues handle the interactive vs. deferred distinction. A session will be set up as either deferred or interactive with a certain precedence. If the session happens to be deferred with a routine precedence, all its messages will go to the deferred routine queue. All interactive routine messages will get serviced from their queue before any deferred routine messages, since the interactive and deferred routine precedence queues are completely separate.

b. Direction of Message Flow

Another distinction between interactive and deferred sessions is the direction of the message flow. While interactive flow will be bidirectional, deferred will be unidirectional only. This difference is somewhat blurred with messages sent over the DDN. In the DDN, TCP sets up a full duplex connection with all virtual circuits, and it would be easy to begin sending messages for both types of sessions as soon as the local connection was established. This can be done even before the destination functional module has been informed that there will be incoming messages for it. The messages would simply arrive at the remote LAN's NC and be sent out over the LAN bus with the destination's address on them. Consider, however, the case where the destination process is initially unable to respond to the interactive messages for one reason or another. The source module must take the time to send three transmissions of the same message before it knows it can abort the session. The source process should not have to wait for the message timeout and retransmission cycle (which could take several minutes), to find out that it will not be able to open a session.

To eliminate the possibility of opening an interactive session when the destination is unreachable, there will be an end-to-end acknowledgement during the session open phase. In other words, before any session messages can be sent, both local and remote modules must be ready to receive.

On the other hand, deferred traffic does not need this handshaking. Traffic is only one way and there is also a capability to deposit incoming deferred traffic in an inactive process's mailbox until he becomes active again. This mailbox capability will be discussed in more detail under the Message Server.

2. Server Processes

NC, like the other functional modules, must take several actions each time it sends or receives a message. One way to handle these message functions is to imbed them in each of the functional modules. This might be a viable alternative in the future; however, there are two main reasons that these functions have been kept separate during the current design of NC. First, these actions are common to all the functional modules, and NC is the first functional module to reach the formal design phase. Keeping the message functions separate will facilitate the design of other functional modules, since the message handling functions need only be developed once. Second, these actions depend a great deal on the specific hardware or operating system of the processor on which the functional module is running. Keeping the message functions separate allows high level interfaces to be built now, while deferring the development of the physical interfaces until the implementation phase.

The above reasoning led to the development of the server process called the Message Server. It is recognized that the actual SPLICE LAN provided by the vendor might implement many of the Message Server's functions. If this occurs, the previously defined interfaces of the Message Server can be mapped directly to those of the vendor's network.

The other server processes that have been defined are Local Communications and Buffer Manager. All three server processes have been described in Chapter 2.

Regardless of their eventual implementation, Message Server, Buffer Manager and Local Communications act as lower level servers for the functional modules, relieving them of the more mundane and trivial matters of message traffic.

Since messages being handled by these server processes must be passed to each of them in turn, at least one copy of the processes will exist in every physical processor. This way they can all work on the same message by simply passing the

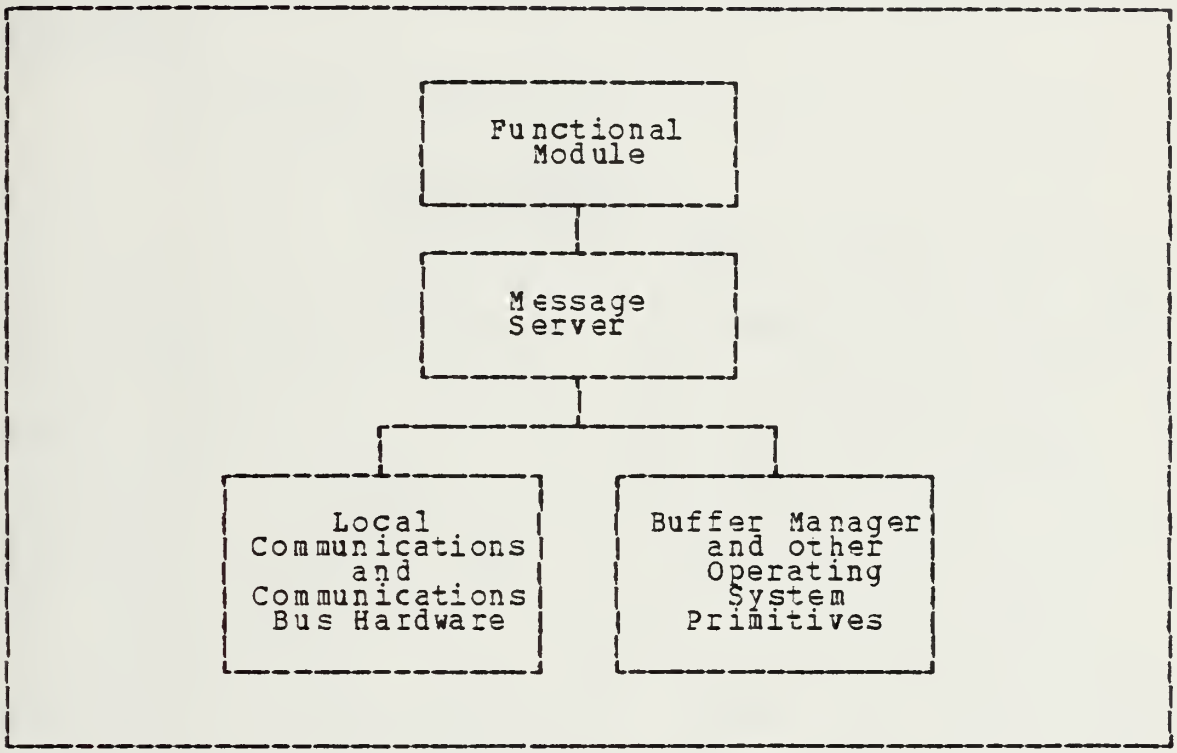


Figure 3.1 Functional Module and Server Process Hierarchy.

address of the message buffer among them. Figure 3.1 shows the relationship among these server processes and the functional modules located within the same processor.

a. Local Communications (LC)

LC was shown as a functional module in [Ref. 1,p.11] because the level of design at that time was such that the actual implementation of LC need not be considered. It is now felt that LC will be the collection of lower level protocols provided by the eventual vendor of

the SPLICE local area network. These protocols might even be provided on a separate microprocessor board within each processor. This makes any interface to LC very system dependent and subject to change. Therefore, Message Server will provide the interfaces necessary to LC, keeping the functional modules separated from implementation dependent functions.

b. Message Server (MS)

(1). Mailbox Manager.

An assumption made in [Ref. 1,p.41] is that the SPLICE LAN will have a mailbox capability. That is, for any deferred session, the message can be deposited into a mailbox and a session will be initiated with the mailbox as the source (and later the destination). To support this function, an MS will have the capability of managing the mailboxes for its functional module. It is important to point out that the absence of mailboxes in the LAN would result in the inability to send anything but interactive traffic, since there would be nowhere to store messages for inactive processes. There will also be times when a destination is unreachable for one reason or another. If SS detects this during the open phase of a session, the session should not be allowed to continue. If the session is a deferred one, there is no reason why the user must be the one to retry later, when MS could do it for him. Suppose however that SS doesn't know that the destination is unreachable and opens the deferred session. The message is sent to the destination, but the destination MS need not throw the message out, it can deposit the message in the process mailbox and cause the process to be notified of mail when it is reachable again.

(2). Session Timeout.

Suppose in another case, an interactive session has been in existance for several minutes without any message traffic. The session is tying up resources that might be needed for other new or more active sessions. To help alleviate this problem, MS will contain a "session timer" that will keep track of when the last message was sent or received. If the elapsed time exceeds a certain limit, MS will issue an abort and notify the user of the action.

(3). Session Independence.

Once a session is set up, there is no need for Session Services to get involved until the session is closed or aborted. For this reason, SS will pass all the vital information to MS at the beginning of the session, so that SS can drop out of the picture and let MS do all the work until the session ends.

c. Buffer Manager (BM)

Since [Ref. 1,p.14] requires that each functional module have enough dedicated memory to handle at least one incoming message, it is important for NC to know how message buffers will be allocated. Since the physical processors for SPLICE have not even been defined yet, it is not known how the future processors will handle memory allocation. Therefore, Buffer Manager was developed as a logical abstraction of the buffer allocation process, so that the functional modules can be designed consistantly, even with underlying hardware unknowns or later changes.

3. Session Services Module

a. Session States

All modules, including NC, use Session Services (SS) each time a session is opened or closed. NC and SS must communicate with each other in order to set up a session. This communication requires that NC and SS go through a series of states in order to control the session traffic. This progression from state to state insures that no message traffic is sent before the proper LAN or DDN connections have been made.

b. Fixed Location

It should be emphasized that SS is the central distributer of addresses for all other functional modules in the SPLICE Network. Because of this, all other modules must know how to get to SS in order to set up a session. Since message traffic to SS will be handled just as any other traffic, all functional modules must know the logical and physical addresses of SS. This requires that SS cannot be physically moved without modifying its address in all functional modules that will want to send messages to it.

c. Session Number

Session numbers are given out by SS in order to identify the messages for that session. These session numbers are unique within each LAN only. There is the provision for one functional module to have several sessions in operation at one time [Ref. 1,pp.42-43]. There is also the possibility that some of these sessions are with local modules and some are with remote modules. It is reasonable to expect that two functional modules will have multiple sessions between them (TM and DBM are likely candidates). Unique message numbers are assigned in each LAN, but there

is no guarantee that a message coming from a remote LAN will have a message number unique to the destination LAN. Therefore, there must be a method by which these modules can uniquely identify incoming messages and acknowledgements. One method would be to have NC keep a table of the message numbers and to which session each number maps. This method would create a lot of unnecessary overhead and might not be possible when several different LANs are communicating with each other. The best solution is the addition of a session number field to the SPLICE message and acknowledgement formats (see Figure 2.1). With the session number in place in every message, it becomes an easy task for NC to map between session number and local connection name for every message it handles. For outgoing messages, NC merely does a table look-up with the local session number as the key and then invokes TCP with the Local Connection Name corresponding to that session number. For incoming messages, TCP will always return to NC with the Local Connection Name as a parameter, thus allowing NC to map to the local session number. See Figure 3.3 for a representation of NC's session table.

4. Network Services Directory

The Network Monitoring Center will continually send updates concerning LAN and functional module status to each LAN's Recovery Management module [Ref. 1,p.28]. RM, in turn, will update the Network Services Directory (NSD). The fields in the NSD [Ref. 1,p.40] consists of only services codes and addresses, but no status information. Adding status information enables SS to determine whether a destination is reachable during the opened phase of a session. This greatly reduces the time involved in determining whether an interactive session can be opened or not. It also gives the capability of developing an online query

function that would inform the user of reachable destinations. The function, which might be called "STATUS", could inform the requester of which entities within the NSD are reachable.

5. Recovery Management

Network statistics are a vital part of monitoring the capacity of a network. Along with the continual network updates, the SPLICE Network Monitoring Center will likely be requesting periodic feedback about the LAN from Recovery Management (RM). Since RM will be reporting the LAN statistics, it is logical that RM also contain the process that gathers these statistics.

C. DDN-RELATED DESIGN ISSUES

1. Implications of Current SPLICE Design

As briefly addressed in Chapter 1, a number of design decisions were made for the SPLICE project prior to the NC design effort that had broad implications for the NC module. These will be addressed first to provide a backdrop for decisions made during the course of the NC design.

a. NC as a DDN Gateway

One of the general ways of examining the NC module is to look at how it serves as a DDN gateway module. To use the term "gateway" is not consistent with ARPANET notion of a gateway [Ref. 14]. ARPANET gateways are normally implemented at the Internet Protocol (IP) level. NC will be a TCP level gateway. TCP/IP will be implemented in the SPLICE FEP, but not elsewhere within the LAN. [Ref. 1,p.18].

When NC receives a call from TCP concerning an incoming message, NC must access the buffer, and using the

LAN message format, find the LAN addressee. Recall that the data portion of the TCP segment is the entire LAN message/ message fragment. When NC calls upon TCP to send a message, it passes as a parameter to TCP the destination (SPLICE FEP) address. The way that NC performs its operations is important because it does not perform routing in the way performed by most gateways. NC depends upon the fact that the local address for the message can be found in the message buffer. Note that when TCP passes the buffer pointer to NC, the buffer contains only the data that was passed to the source TCP. That data is a LAN-formatted message sent by the source NC.

NC's gateway design will initially mean that SPLICE LANs can only send or receive messages to or from other SPLICE sites who utilize the same message format and handling procedures. It is possible that message communications with non-SPLICE activities on the DDN might be implemented later. The major elements needed for this type of internetworking are standardized message formats and common handling procedures (acknowledgements, session practices, etc.). A DDN standard for internetworking of LANs utilizing TCP-level gateways might be developed at a later date. SPLICE adoption of such a standard could lead to greater intercommunication with non-SPLICE communities on the DDN.

b. NC Assembly and Disassembly of Messages

Any given NC module will only receive messages from other SPLICE LANs, and the message format and maximum message or message fragment size will be common to all LANs. TCP is assumed to be capable of accepting an outgoing message (buffer) for a maximum size SPLICE message or message fragment. TCP will disassemble the LAN message into TCP segments before sending it onto IP for datagram service to the destination. At the destination, TCP reassembles the

LAN message putting it back into the exact format that the source TCP received from the source NC. Therefore, NC will have no requirement to disassemble or assemble outgoing or incoming messages. As described in the last chapter, NC must, however, sequence the messages using the stop and wait acknowlegement scheme.

D. DDN FUNCTIONAL REQUIREMENTS IMPOSED UPON NC

1. Precedence and Security

A basic DDN subnet requirement is that

> "...there be reliable mechanisms for assigning network resources for different user priority levels such that the higher priority users are assigned the use of scarce resources ahead of lower priority users." [Ref. 4,p.102]

The precedence mechanisms throughout the DDN depend upon the correct information in the precedence field of the IP header on DDN messages [Ref. 15,p.12]. Precedence information is given to IP by TCP, and TCP will use default values if precedence is not specified by NC. Similarly, the DDN must be capable of handling messages of different classifications and does so, in part, through the use of a classification field in the message header at both the TCP and IP level. TCP will use default values for classification if not provided by NC.

There are basically three alternatives:

- Do not implement classification or precedence capabilities internal to the SPLICE LAN.
- Implement both classification and precedence handling in SPLICE.
- Implement either classification or precedence handling.

If SPLICE chooses not to implement all of its DDN message traffic will traverse the DDN backbone with the

lowest precedence: ROUTINE. This would be highly undesirable during wartime conditions when supply actions such as emergency requisitions may fully justify higher precedences including FLASH. It is essential that SPLICE LANs have the capability to utilize the precedence mechanisms of the DDN. With regard to security, it is true there is no present need for a classification field on the LAN message format; however, if the need later evolves for SPLICE, and classified systems are added, then changes would have to be implemented to support a new message format. Implementation at a later (operational) stage of SPLICE would be far costlier than providing for that classification capability now. The addition of two extra fields to the message format will result in negligible overhead to SPLICE. Recall that a user will establish his precedence and security levels at the outset of a session, and these remain fixed throughout a given session. Default values can be used for most sessions.

Classification and precedence fields will be added to the SPLICE message format. Procedures to support proper handling of classification and precedence will be integrated into all applicable SPLICE modules.

2. TCP STATUS Command

The TCP command named STATUS provides status information on the particular local connection requested [Ref. 6,p.49]. This is an optional command which means its implementation is not required by the vendor implementing TCP. The information returned from TCP may include the items shown in Figure 3.2.

With regard to NC, a decision must be made about whether to implement routines to utilize the TCP STATUS command. If NC does provide the routines to utilize the STATUS command, and the vendor does not implement them in TCP, the routines would have to be modified to avoid a TCP

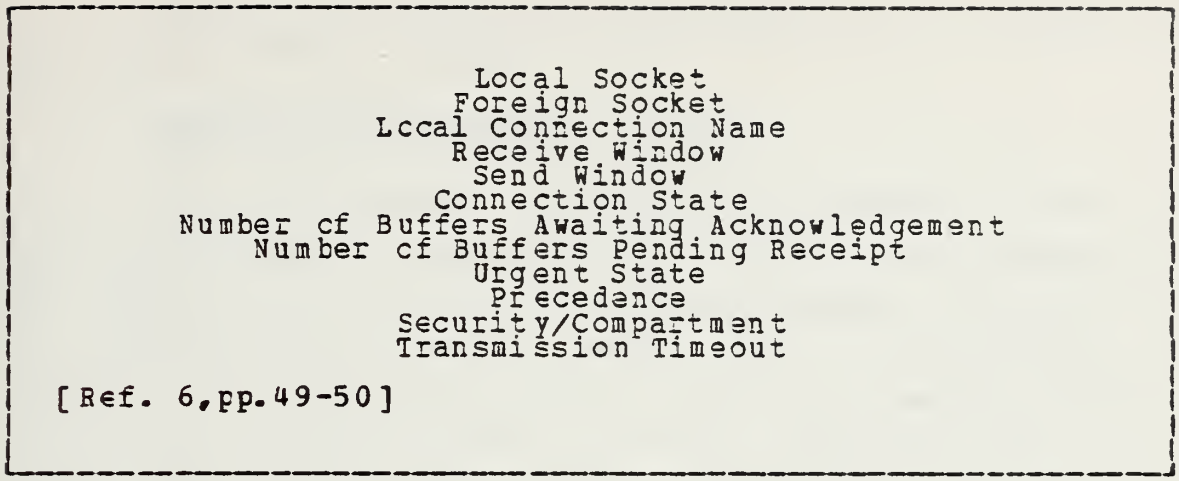


Figure 3.2 Information Returned by TCP STATUS Command.

error. If the vendor implements and NC provides access, then an additional amount of overhead is levied on NC and TCP every time the command is utilized. On the other hand, if NC does not implement but the TCP vendor does, the user will not have the benefit of the information. Also, there is the advantage of no additional overhead induced on NC or TCP if the command is not supported.

One may also argue how useful the STATUS information is to the normal SPLICE user. While the information may be useful for debugging purposes or for personnel maintaining the computer system, Almost all of the data shown in Figure 3.2 will not be wanted by the typical SPLICE user. More often, a user will be interested in whether a particular destination is "up" or reachable via SPLICE. As discussed earlier, a query capability should be added to Session Services so that users can check the Network Services Directory (NSD) for user information. Similarly, a user may query to obtain information about his particular session. Such a query capability would not detract from the performance of TCP or NC.

NC will not implement the routines to utilize the optional TCP STATUS command.

3. Message Retransmission by NC

The SPLICE design requires NC to employ a stop and wait acknowledgement scheme. Under this method NC sends, via TCP, each outgoing message or message fragment and waits for an acknowledgement before sending the next message in sequence. Underneath NC, TCP has taken the message provided by NC and attempts to send it to the destination TCP. The source TCP employs a retransmission scheme and will continue to retransmit, periodically, the NC message until a destination TCP acknowledgement is received, or the time to live, associated with the message, expires. The stop and wait acknowledgement scheme [Ref. 1,p.34], did not specifically require NC to perform retransmissions.

A decision must be made whether NC should implement the routines to perform retransmissions. If NC performs retransmissions, it will effectively be duplicating the work of TCP, will be a source of duplicates, and will also tend to slow TCP down. Assuming that TCP is operable, TCP will be performing retransmissions on a specific message at the same time that NC would request TCP to send the same message again. If TCP cannot get an acknowledgement from the destination TCP, then it is not possible that NC can get an acknowledgement from the destination NC. The connection should be abandoned and the destination assumed to be in an erroneous state. If, in fact, the source TCP is not properly functioning, then NC should ABORT anyway and not attempt any retransmission. In general, retransmission should not be performed at multiple levels. Analogously, NC or MS should not perform retransmissions within the LAN if LC routinely performs retransmissions.

NC will not perform retransmission of a message when no acknowledgement is received.

4. TCP Parameters: URGENT, Options

The TCP specification requires the implementation of TCP routines to support the "Options" and the "URGENT" parameters that may be optionally used by NC when invoking TCP with OPEN or SEND commands. The URGENT parameter is used

> "...to stimulate the receiver to process the urgent data and to indicate to the receiver when all currently known data has been received." [Ref. 6,p.41]

Thus, the URGENT contruct allows the user to send data of a more "urgent" nature in the "middle" of a data stream or session of non-urgent data. This may be very important for real-time system users of the DDN, but no such need is dictated by the SPLICE requirements. The "Options" parameter currently allows three alternative uses: End of Option List, No-Operation and Maximum Segment Size [Ref. 6,p.18]. The use of the parameter is optional, and is not needed for any known SPLICE requirement.

E. SUBMODULES OF THE NATIONAL COMMUNICATIONS MODULE

This section serves as a summary of the previously discussed design decisions as they have been integrated into the NC module.

1. General Structure

The general structure of the NC submodules was largely determined by two factors. First, the assumption that SPLICE will provide for interactive and deferred traffic, and second, that TCP progresses through various states as it processes messages through the DDN.

The differences between interactive and deferred message traffic have been discussed earlier, and will not be dealt with again here. It is sufficient to say that these differences create a natural division of functions by which part of the structure of the NC submodules has been determined.

The details of the TCP states have also been discussed previously. The TCP states provide a framework for further division of the deferred and interactive portions. Since NC must deal with the TCP states, NC was designed to deal with only one state at a time. This way, the states of TCP are controlled by the states of NC (see Figure 2.3).

2. Data Structures

Two general data structures emerged during the NC design. The first is the queue. There are three separate sets of queues used in the NC submodules. A graphic illustration of these queues can be found in Appendix C, figure C.1.

The first set of queues consists of one queue for each open session. This is a result of the required NC to NC acknowledgement for each session. NC must queue up any waiting messages in a certain session until an acknowledgement is received from the remote NC for a previously transmitted message in that session. This first set of queues funnels messages to the other two sets, never allowing more than one unacknowledged message per session at any given time. Of course, there may be many unacknowledged messages between any given pair of NCs, but only one per session. This provision imposes no restriction on the possibility of having multiple sessions outstanding between a pair of functional modules.

The other two sets of queues enforce precedence--one set is for interactive messages, the other for deferred messages. These queues are necessary because of the distinction between interactive and deferred sessions. The precedence queues insure that the session message with the highest priority is given to TCP first--with the interactive messages given a higher priority than deferred messages of the same precedence. TCP has mechanisms to handle precedence, but does not distinguish between interactive and deferred sessions.

NC does not queue any messages sent through Message Server, since MS already has precedence queues in it.

The second data structure present is the session table. Its structure is shown in Figure 3.3. The session number and local connection name entries are used to map the

Remote NC Address	Session #	Loc Conn Name	Sent Last	Received Last

Figure 3.3 National Communications Session Table.

messages to their proper destinations. The remote NC address is necessary for opening and closing a session, when the two NC's will be communicating directly.

3. General Functional Flow

In summary, there are essentially three factors that determine how a message is treated by NC.

1. Whether the message is incoming from or outgoing to the DDN.
2. Whether the message is interactive or deferred.
3. The particular state in which the session is operating.

These three factors are well pronounced in the algorithmic description of NC (Appendix C). The main module of NC (NC_MAIN) examines the messages coming from Message Server or TCP and determines which submodule to call. When the proper submodule is called, it will use the session number or LC Name on the message to map it to the proper destination. The submodules themselves handle NC-NC level acknowledgements.

The Error-Handler is invoked when TCP errors are passed to NC. Almost all TCP errors will result in the session being aborted and the user notified as such. Detection of any LAN errors will be the responsibility of the Message Server.

IV. NC IMPLEMENTATION ISSUES

The present NC design has been detailed to the level of a broad algorithm, and should provide an implementor with a general "blueprint" for an executable process or series of processes. This chapter is meant to express some issues that will need to be resolved or dealt with to fulfill the design. Indeed, the resolution of some of these issues may require a careful restructuring of the design.

A. A MULTITASKING ENVIRONMENT

The present design has primarily dealt with a sequential model of the execution of NC. As discussed in Appendix B and elsewhere, it is believed that the design structure can be mapped into various structures that support multitasking. Given the stop-and-wait acknowledgement scheme, the handling of precedence and the time delays associated with DDN traffic, NC should be heavily oriented towards concurrency mechanisms. Far greater message processing efficiency may be obtained if NC services or multiplexes several sessions concurrently.

As detailed in [Ref. 7], obtaining protocol efficiency may also require an implementation to rely heavily on the FEP operating system; however, this may not be desirable from a software maintenance standpoint, as changes or new versions of the operating system may require modification of NC.

Another consideration involves how TCP/IP will be supported. As will be addressed shortly, TCP/IP may be implemented on an "outboard" microprocessor. An outboard microprocessor is a dedicated microprocessor executing

TCP/IP that may be installed within the FEP and it would probably communicate with NC via the internal data or control bus of the FEP. This may greatly reduce the DDN protocol overhead in the SPLICE FEP, and NC and its Message Server could support an even higher number of concurrent sessions. This may, in turn, produce a bottleneck for access to and from the outboard computer. In short, good performance in a multitasking environment is likely to require considerable tuning.

B. TCP/IP IN AN OUTBOARD COMPUTER

As mentioned in the previous section and in Chapter 1, one probable implementation of TCP/IP may be an outboard computer to the SPLICE FEP. Microprocessor-based Host Front Ends (HFE) are likely to be commonplace on the DDN and some preliminary work has already begun on such an implementation by the Naval Telecommunications Automation Support Center (NAVTASC). This type of TCP/IP HFE should off-load the SPLICE FEP and thus provide more resources to the resident processes of the FEP. One can also anticipate the possible disadvantages of this approach. The Input/Output ports of the FEP can be potential bottlenecks to the overall throughput of the LAN to DDN interface. Some implementations may

> "...end up copying the data because there is no shared memory between processors, and the data must be moved from process to process via a kernel operation. Unless the amount of this activity is kept strictly under control, it will quickly become the major performance bottleneck." [Ref. 7,p.11]

One final point is the requirement to provide an interface between NC and TCP. Although this will probably entail the use of the operating system (kernels) and the communications bus between processors, another layer of interface

routines and calls will have to be built into NC, the operating system or another separate process. Such interfaces can also lead to possible bottlenecks.

C. LANGUAGE ISSUES

As various SPLICE module designs approach the point of actual implementation, the choice of implementation language must be carefully made. The SPLICE Request For Proposal (RFP) allows for either Pascal or Ada for the system programming, and COBOL for the applications modules.

For a number of reasons, any use of Ada for implementation in the near term is probably ill-advised. The language was found to be a superior System Design Language (SDL), and a number of government contractors have already embraced its use as a Programming Design Language (PDL); however, it may be a year or two until good production compilers and other essential tools are available. More importantly, a lot of questions concerning Ada's run-time behavior remain unanswered. At least one study [Ref. 16], has shown that Ada has good potential as a communication systems programming language.

The use of Pascal is certainly less risky, but future implementation efforts must be concerned with the potential disadvantages of the language. As stated previously, the run-time environment of SPLICE should support a high level of multitasking. It is highly desirable that the version of Pascal used contains built-in synchronization primitives. This would eliminate the need to use the operating system primitives, and consequently improve the machine independence of SPLICE modules.

Another important requirement concerns memory management. Most Pascal implementations generate a fatal, run-time error when run-time storage is exceeded. Such errors

must not be fatal in SPLICE and exception handling routines will have to be implemented in the code if not provided by the particular version of Pascal. Finally, selecting a good Pascal implementation must also entail a detailed look into the programming support environment usable with the language.

D. INCREASING EFFICIENCY

There are several reasons why one would want to increase the efficiency in message passing. The reason that should be paramount in the SPLICE effort is to enable the interactive user to get the fastest turnaround possible.

It is recognized that any interface with TCP involves a certain amount of overhead. In this regard, NC should be made as optimal as possible. Some areas in which the current design might be improved include acknowledgement schemes, modularity issues and message buffering.

In addition to improvements within NC, there may be areas on the LAN side in which a different approach could improve message throughput. Areas of consideration include Message Server placement and the distribution of Session Services functions.

E. FOLLOW-ON EFFORTS

The authors have attempted to provide a sound basis for future SPLICE efforts at NPS through this thesis. It is recommended that follow-on efforts should be immediately focused at developing the server processes, initially tackling the Message Server. MS is the key to the LAN's message traffic and once developed, will provide a solid foundation on which to build the functional modules. Buffer Manager would probably have to be built concurrently, but this is considered to be a much easier task than MS.

Once MS is developed, the next logical step is to build TM and perhaps DBM with a deferred capability, only. If the development of NC is completed during this time also, the basic SPLICE Network could be tested, adding interactive traffic later.

All of this developmental effort would be greatly facilitated by the use of a proper testbed. There is an excellent opportunity for the use of the VAX configuration supported by the Computer Science Department at NPS. The department has recently installed an Ethernet capability connecting the VAX machines with a set of microprocessors. In addition, TCP/IP implementations are available for the VAX at a relatively low cost. Using the Ethernet as the Local Communications process and with TCP available, one can envision a highly respectable testbed for the SPLICE module development. Once the LAN is functioning in a deferred mode, the interactive traffic capabilities could be added.

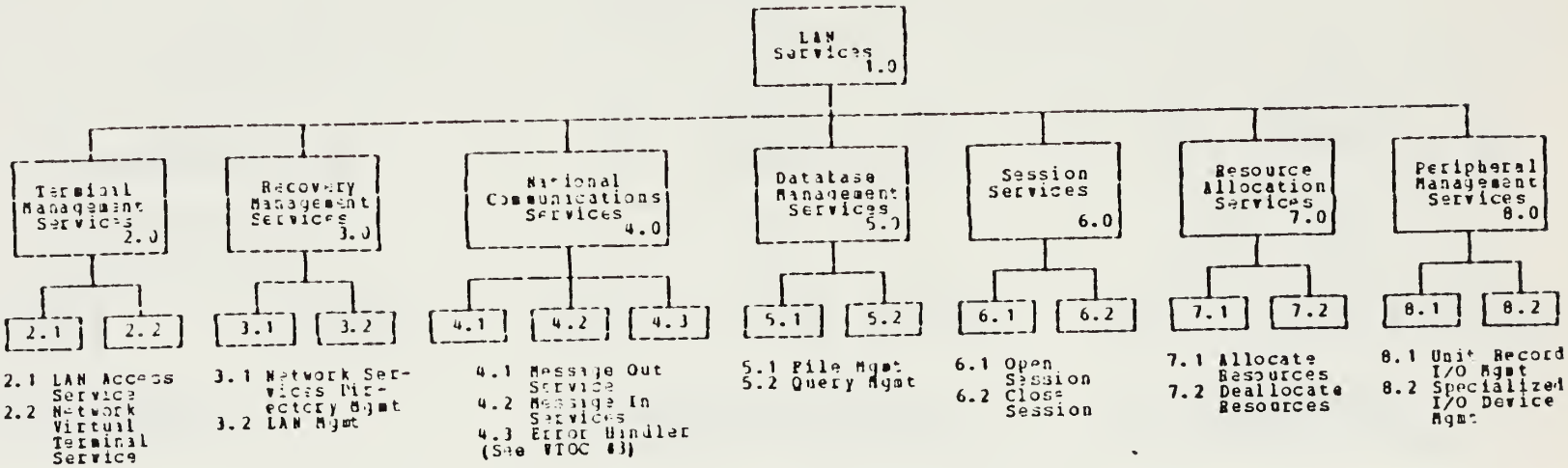
APPENDIX A
HIPO CHARTS

This appendix follows the HIPO conventions used in [Ref. 9] and [Ref. 11]. The first few pages consist of Visual Tables of Contents (VTOC), and are followed by the HIPO charts themselves. The charts are sorted in ascending numerical order; first the functional modules, then the support modules. The LAN modules other than National Communications have been included for the sake of completeness. Only the NC module has been expanded beyond the first three levels on the VTOC.

During the development of these charts, it was found that concurrency was very difficult to represent, since HIPO was designed for sequential processes. However, the HIPO method has proved very useful in the functional decomposition of the modules.

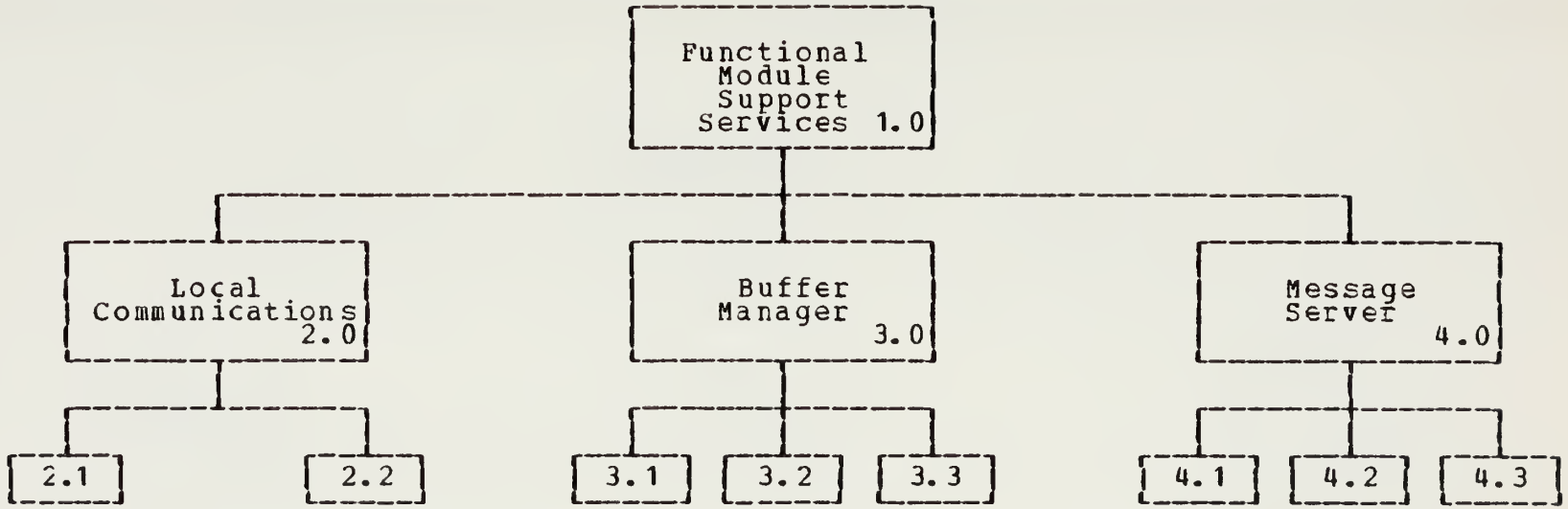
SPLICE Local Area Network

Visual Table of Contents No. 1



SPLICE Local Area Network Support Modules

Visual Table of Contents No. 2



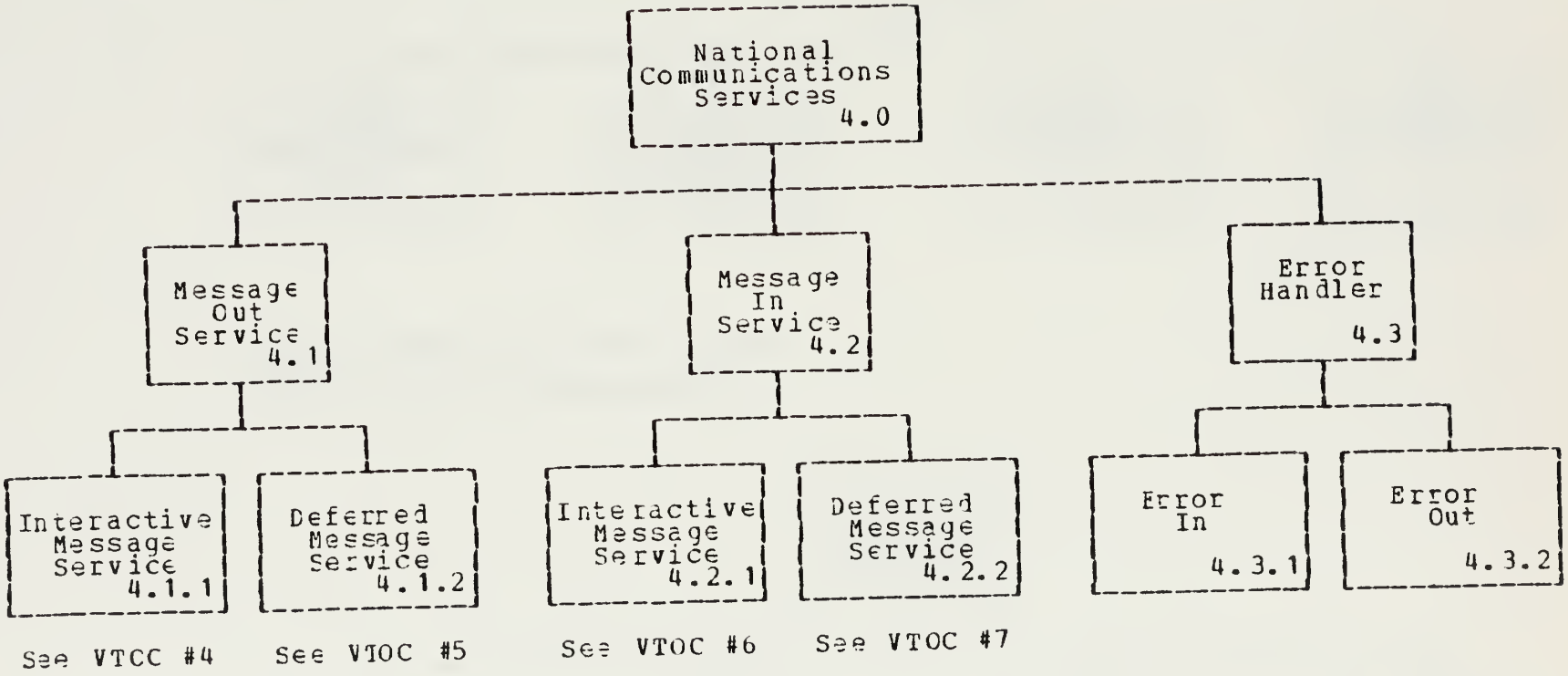
2.1 Message In
2.2 Message Out

3.1 Allocate Buffer
3.2 Deallocate Buffer
3.3 Request More Storage

4.1 Message In
4.2 Message Out
4.3 Mailbox Manager

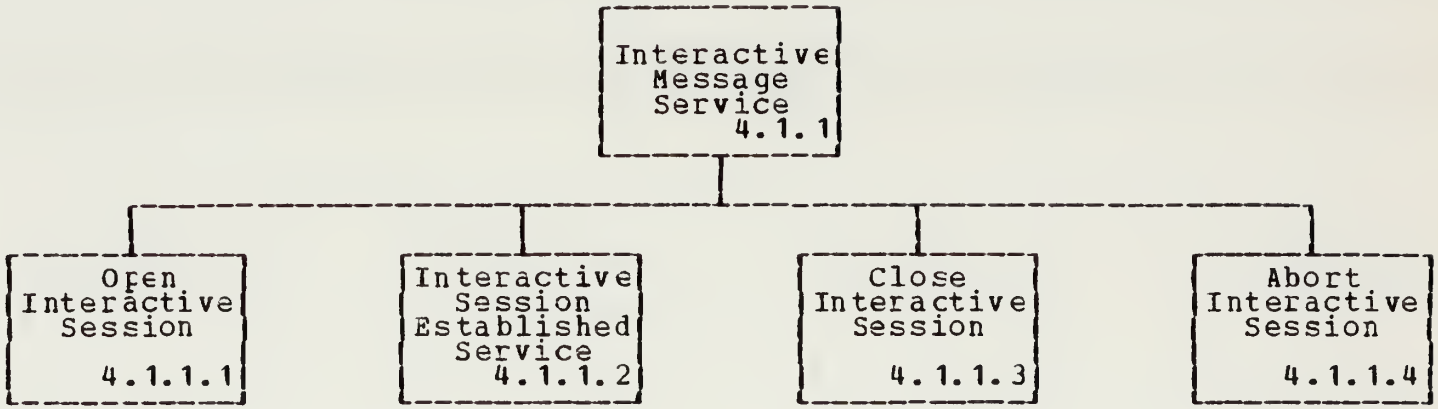
National Communications Service

Visual Table of Contents No. 3



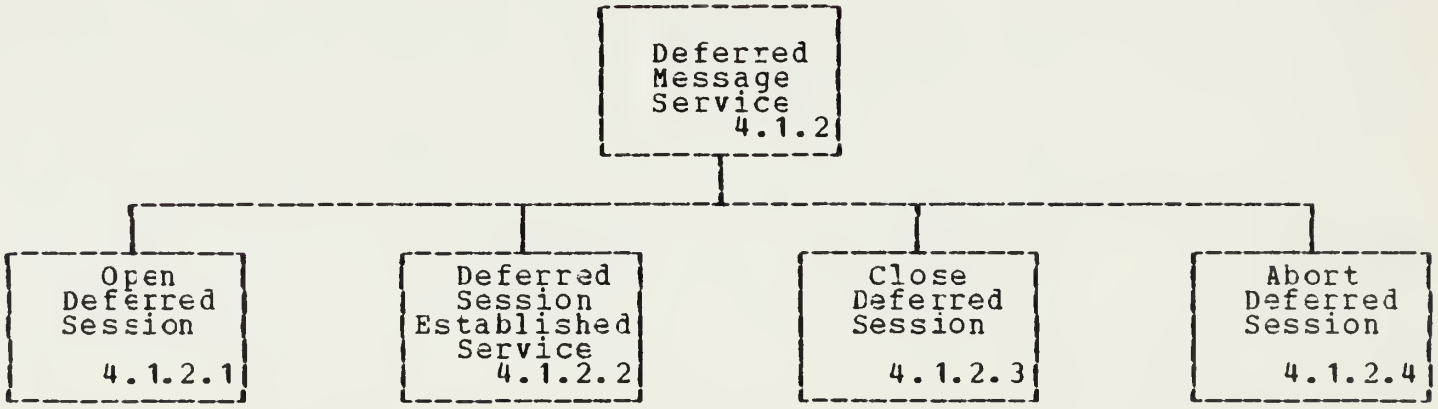
Interactive Message Service (Out)

Visual Table of Contents No. 4



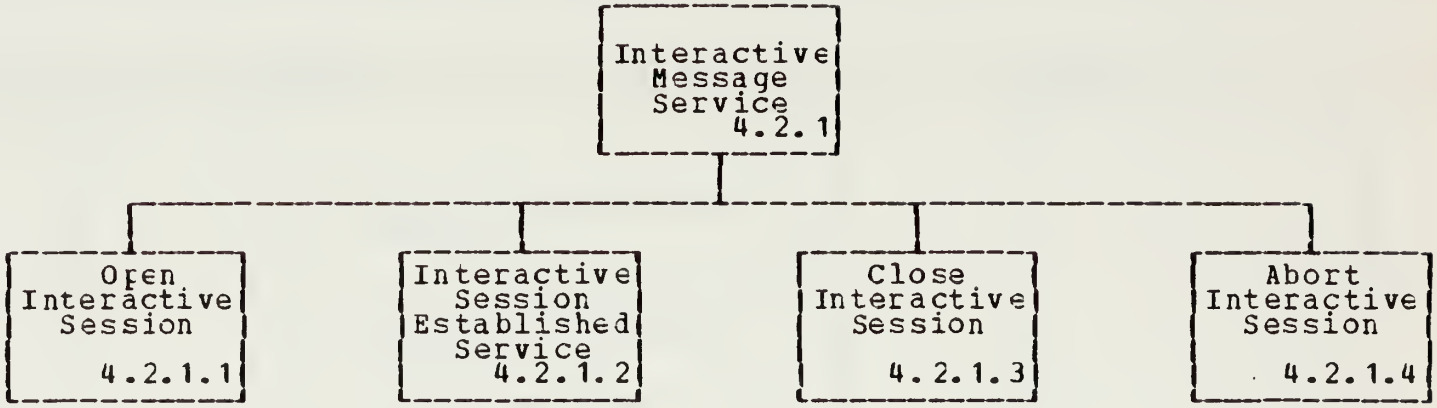
Deferred Message Service (Out)

Visual Table of Contents No. 5



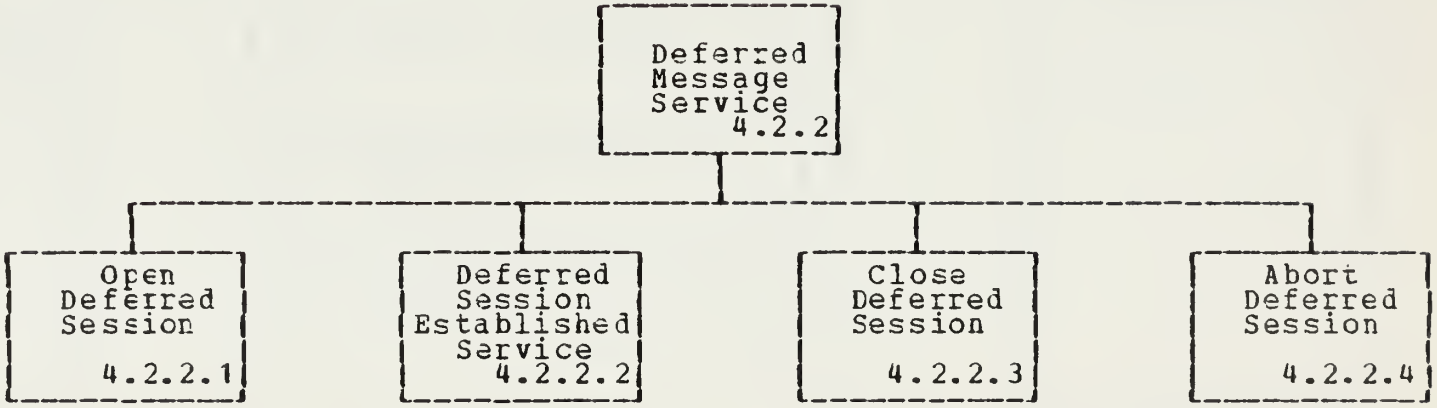
Interactive Message Service (In)

Visual Table of Contents No. 6

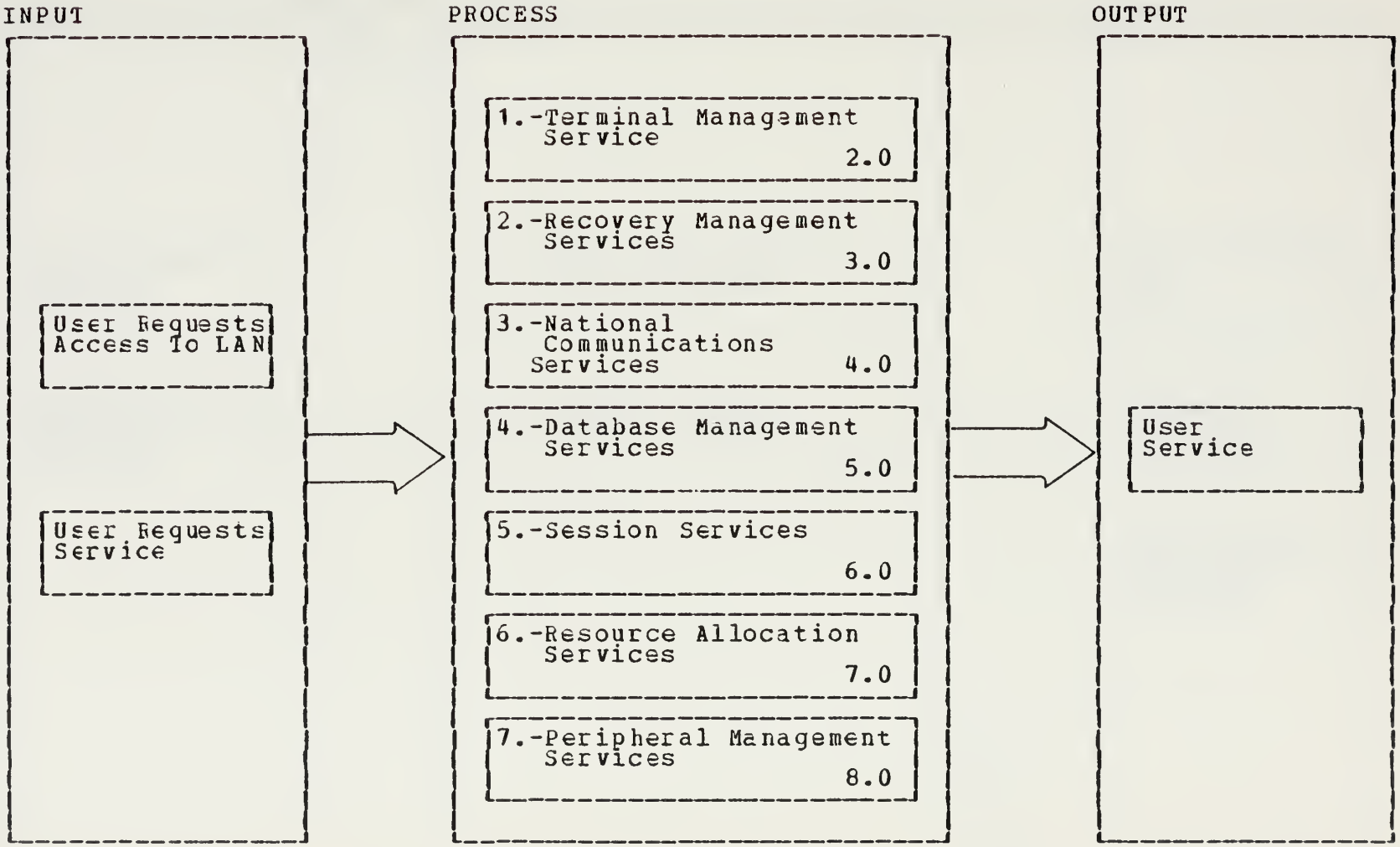


Deferred Message Service (In)

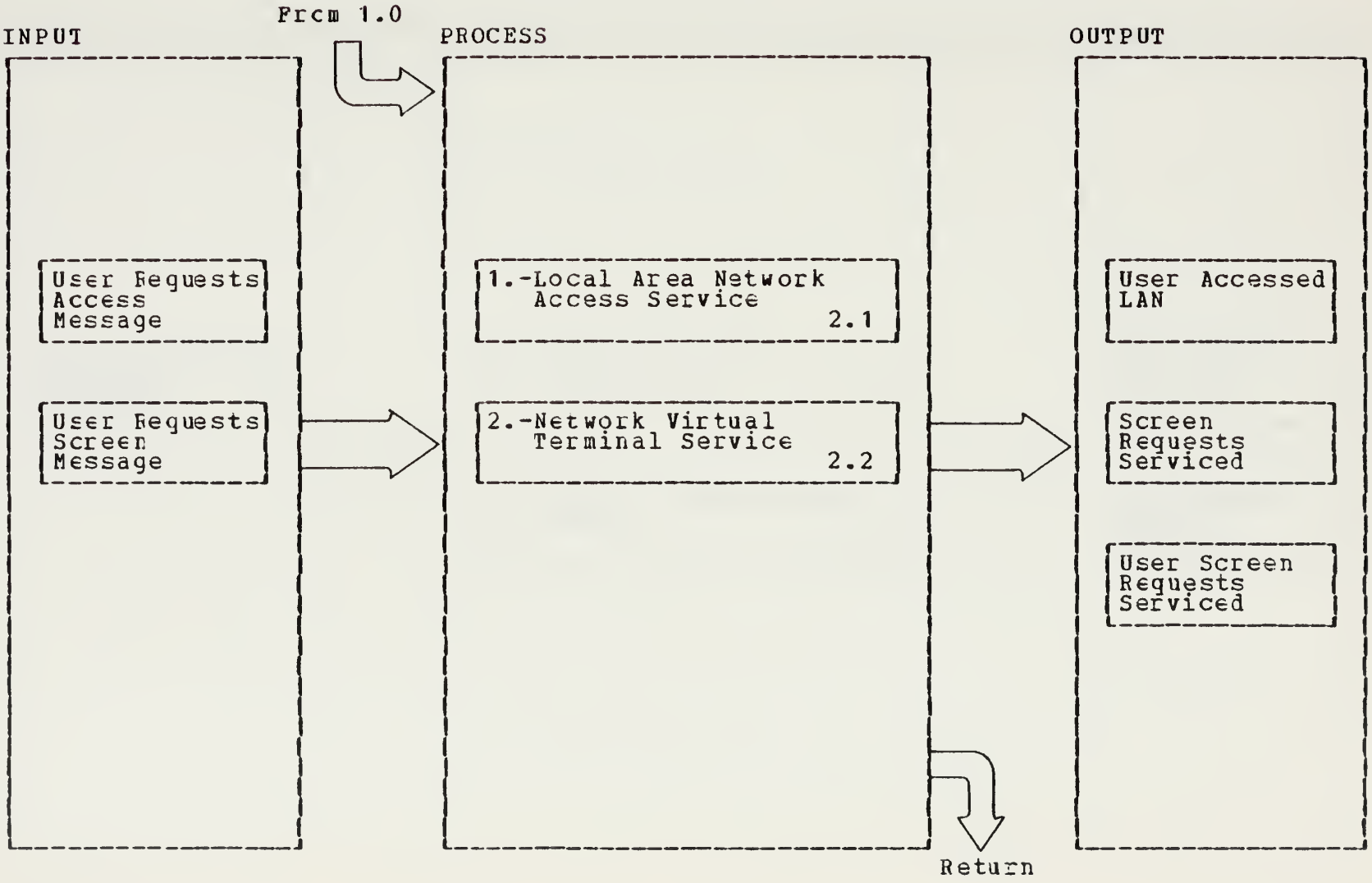
Visual Table cf Contents No. 7



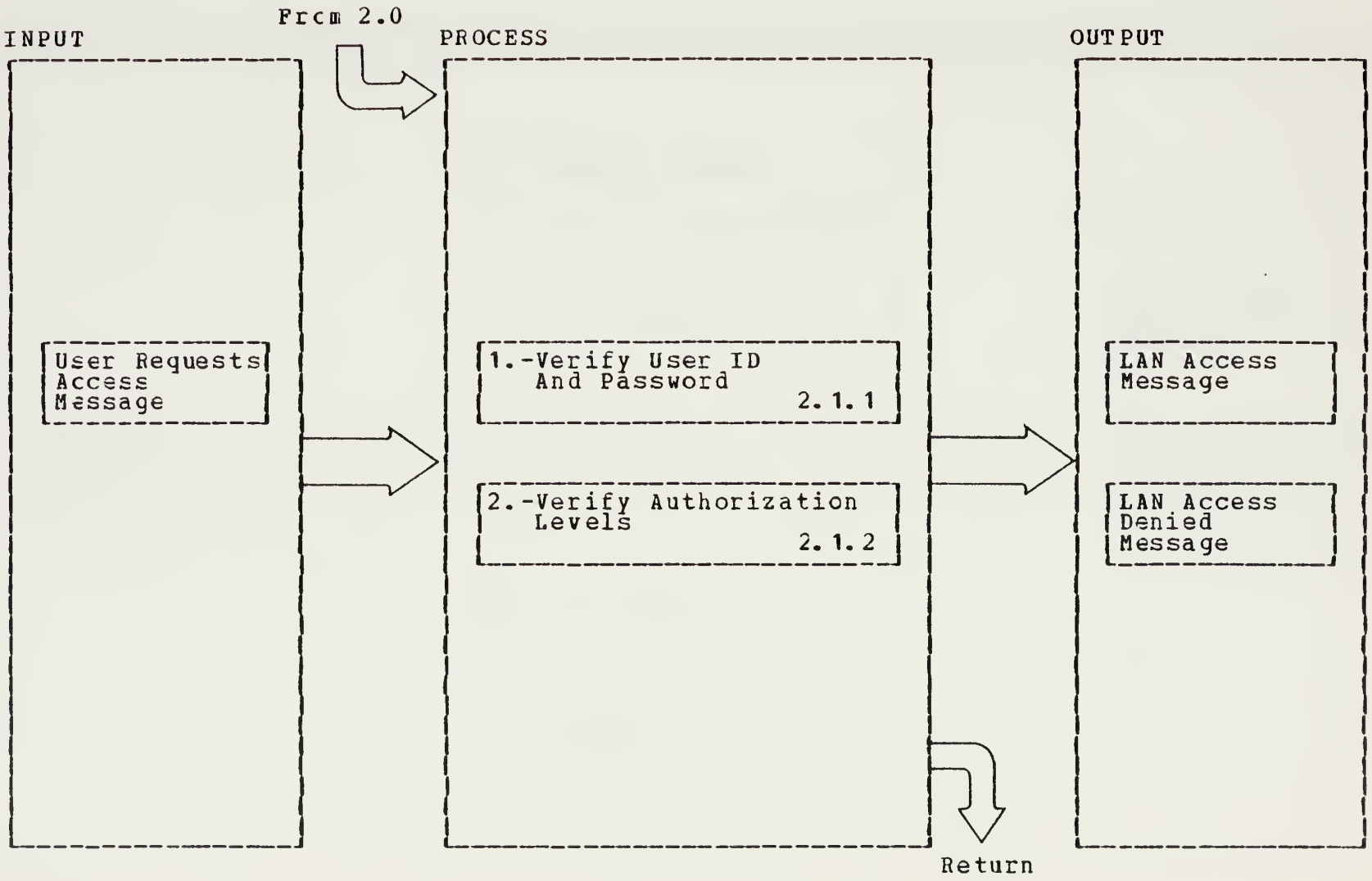
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 1.0 Name:LAN001 Description:LAN Services



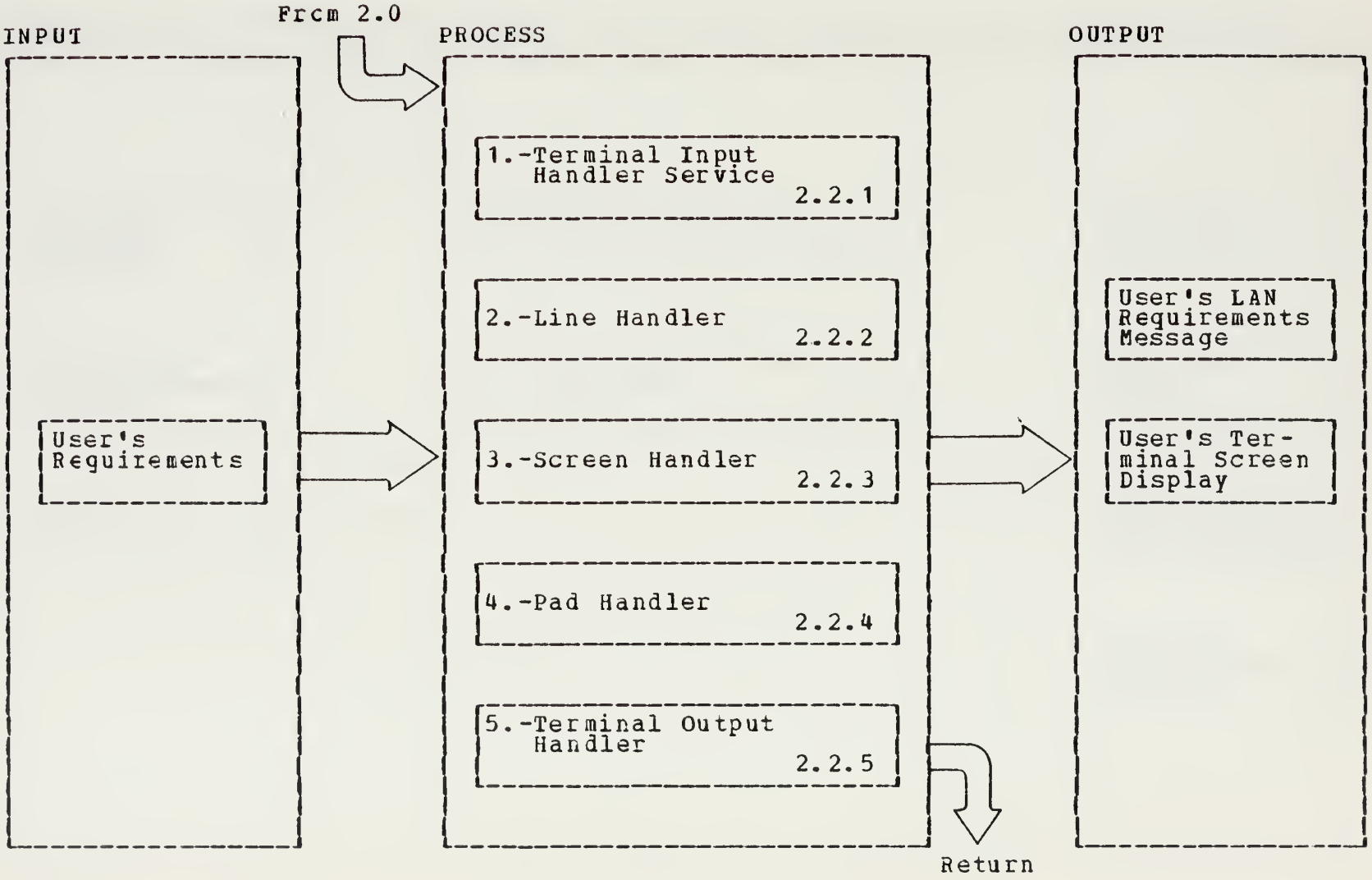
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 2.0 Name:LAN002 Description:Terminal Management Services



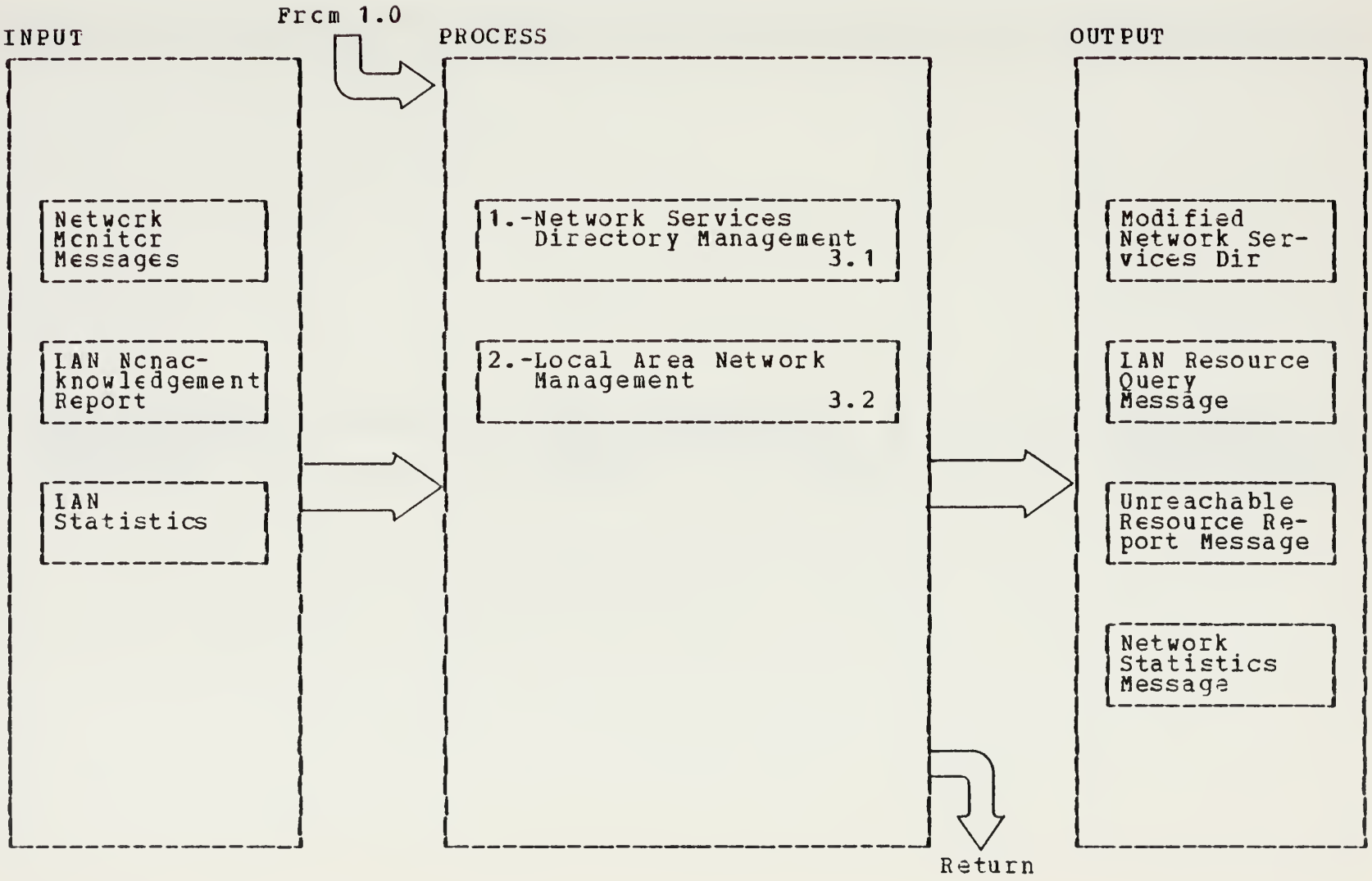
Author:Krebill/Carlsen Syestm/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 2.1 Name:LAN003 Description:LAN Access Service



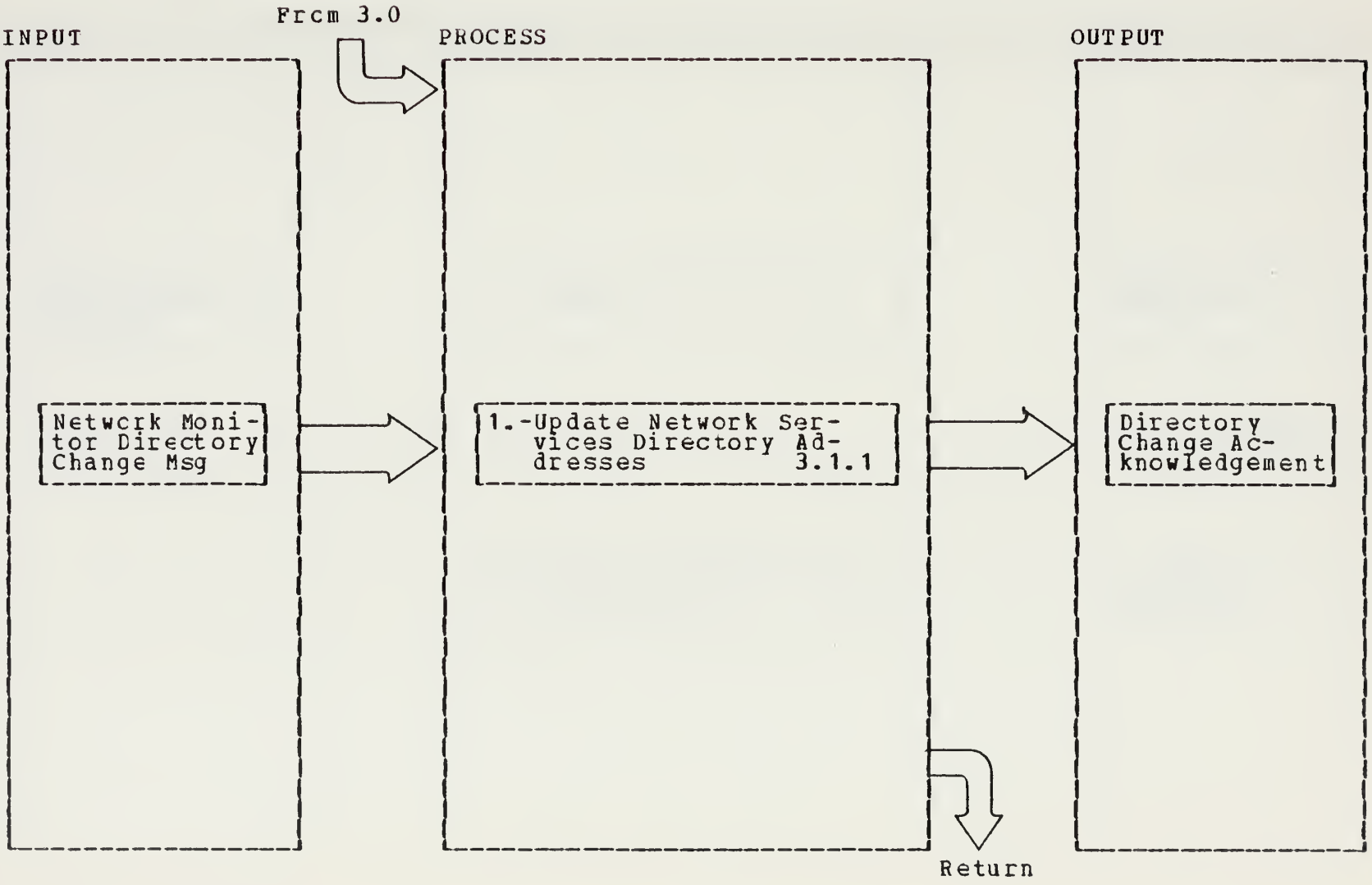
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 2.2 Name:LAN004 Description:Network Virtual Terminal Service



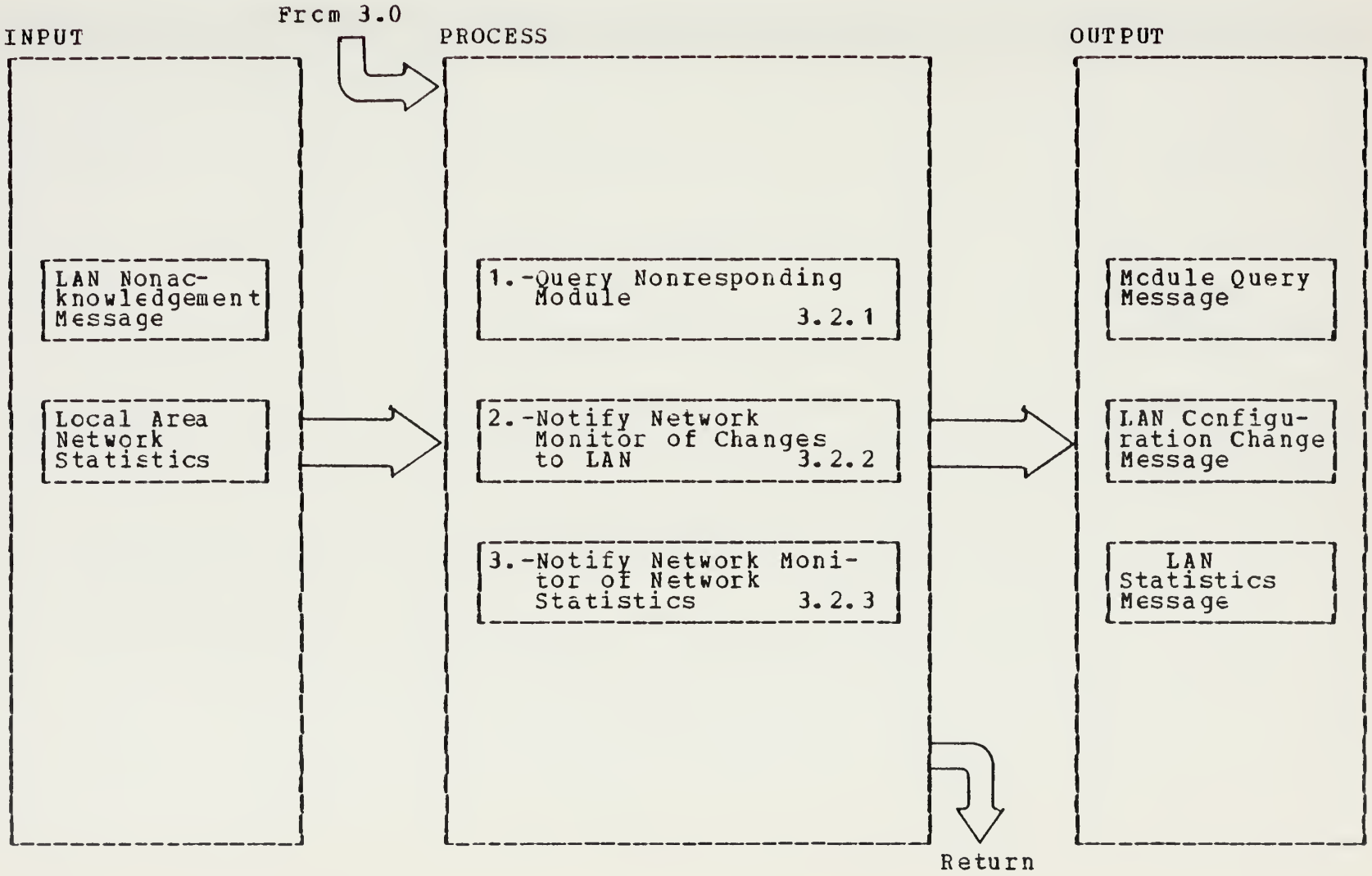
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 3.0 Name:LAN005 Description:Recovery Management Service



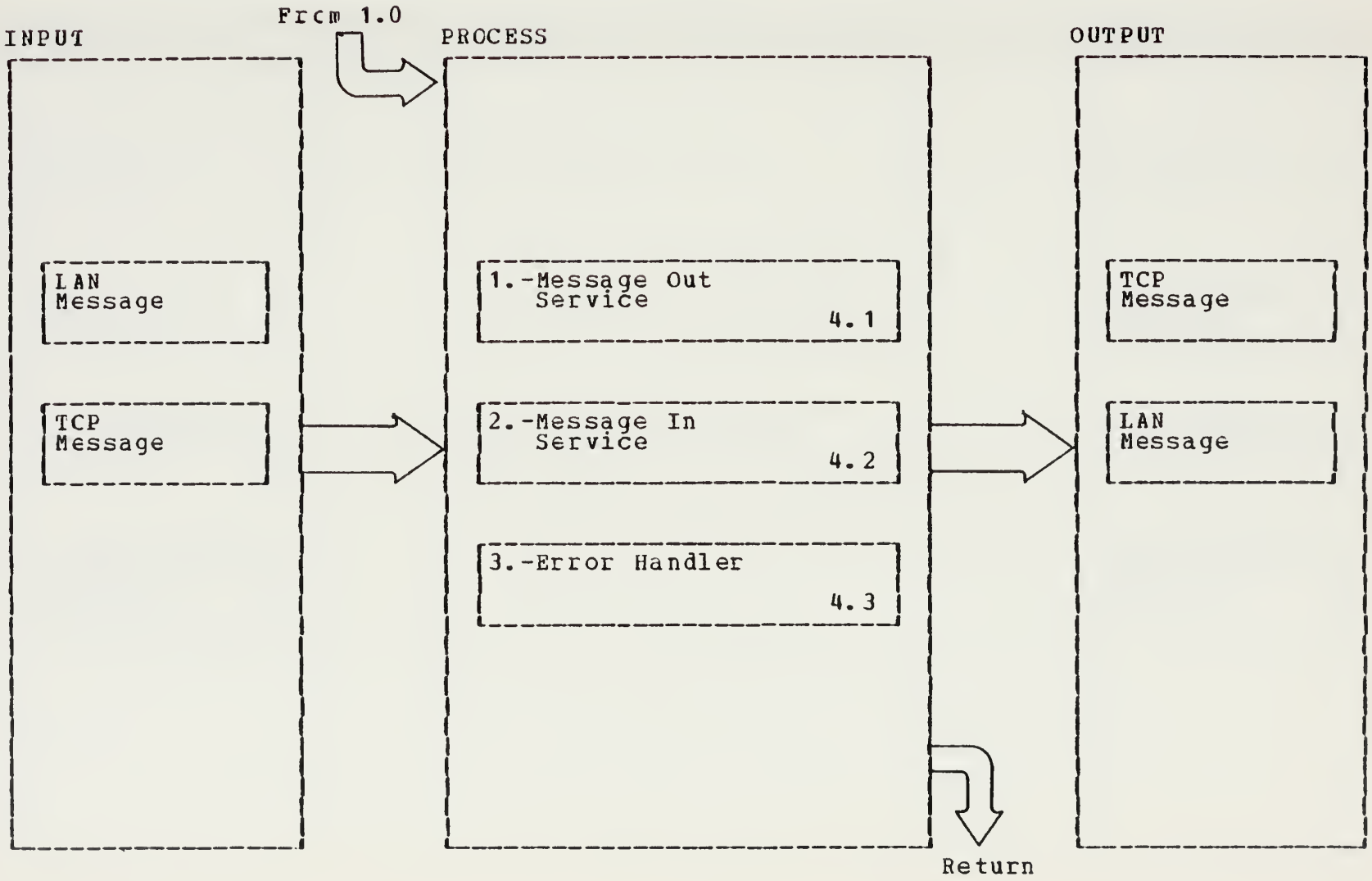
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 3.1 Name:LAN006 Description:Network Services Directory Management



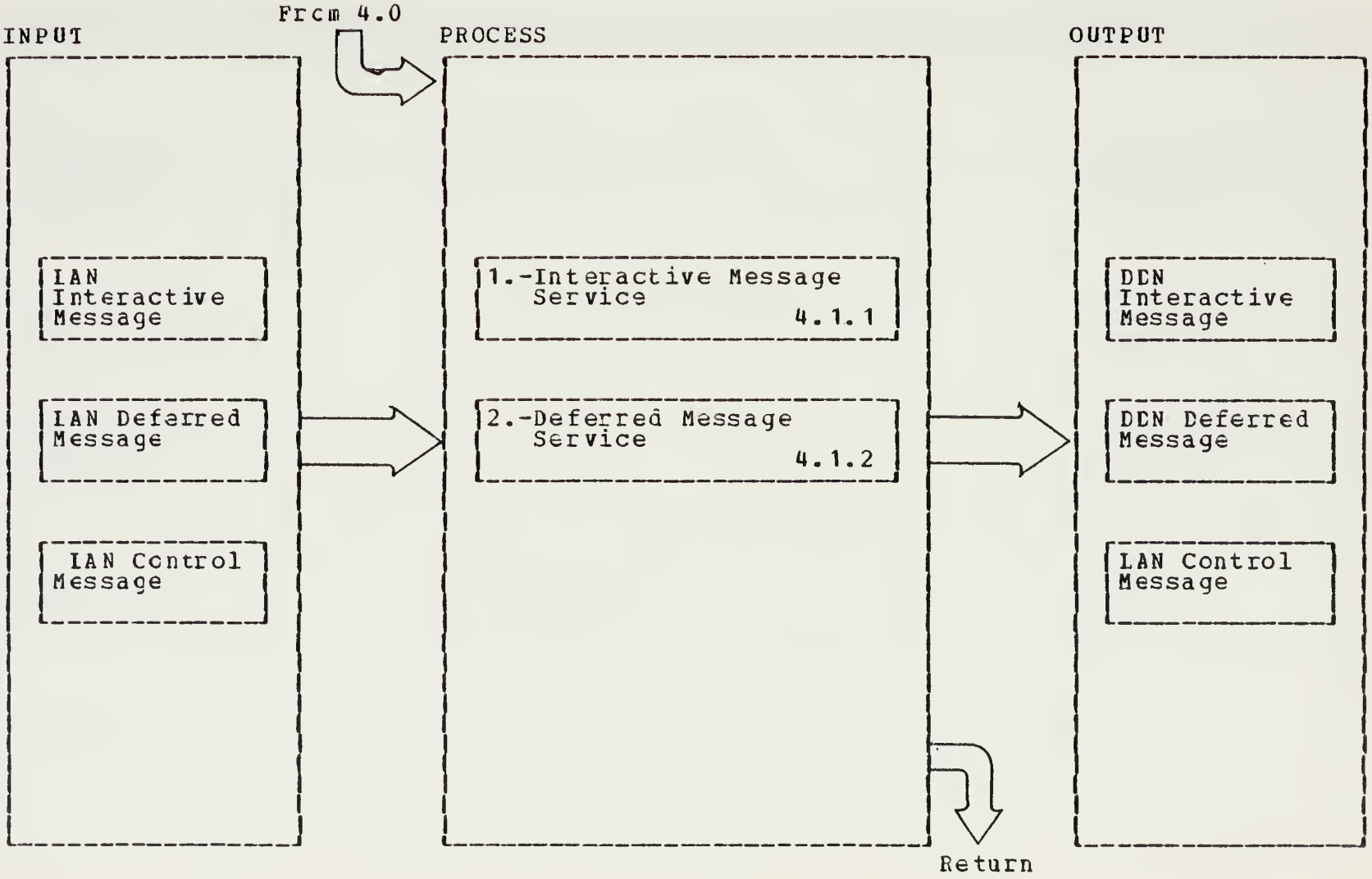
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 3.2 Name:LAN007 Description:LAN Management



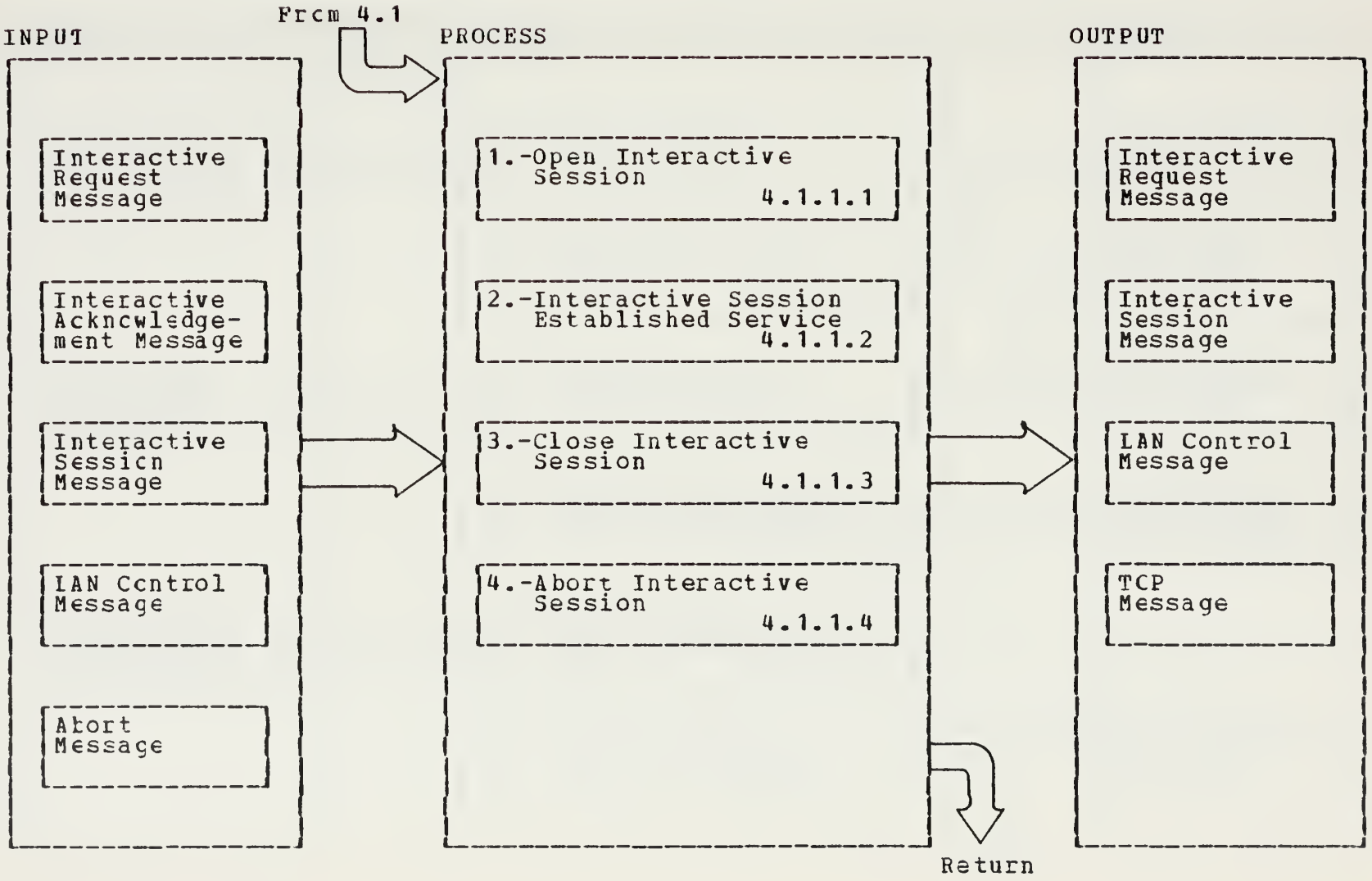
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.0 Name:LAN008 Description:National Communications Service



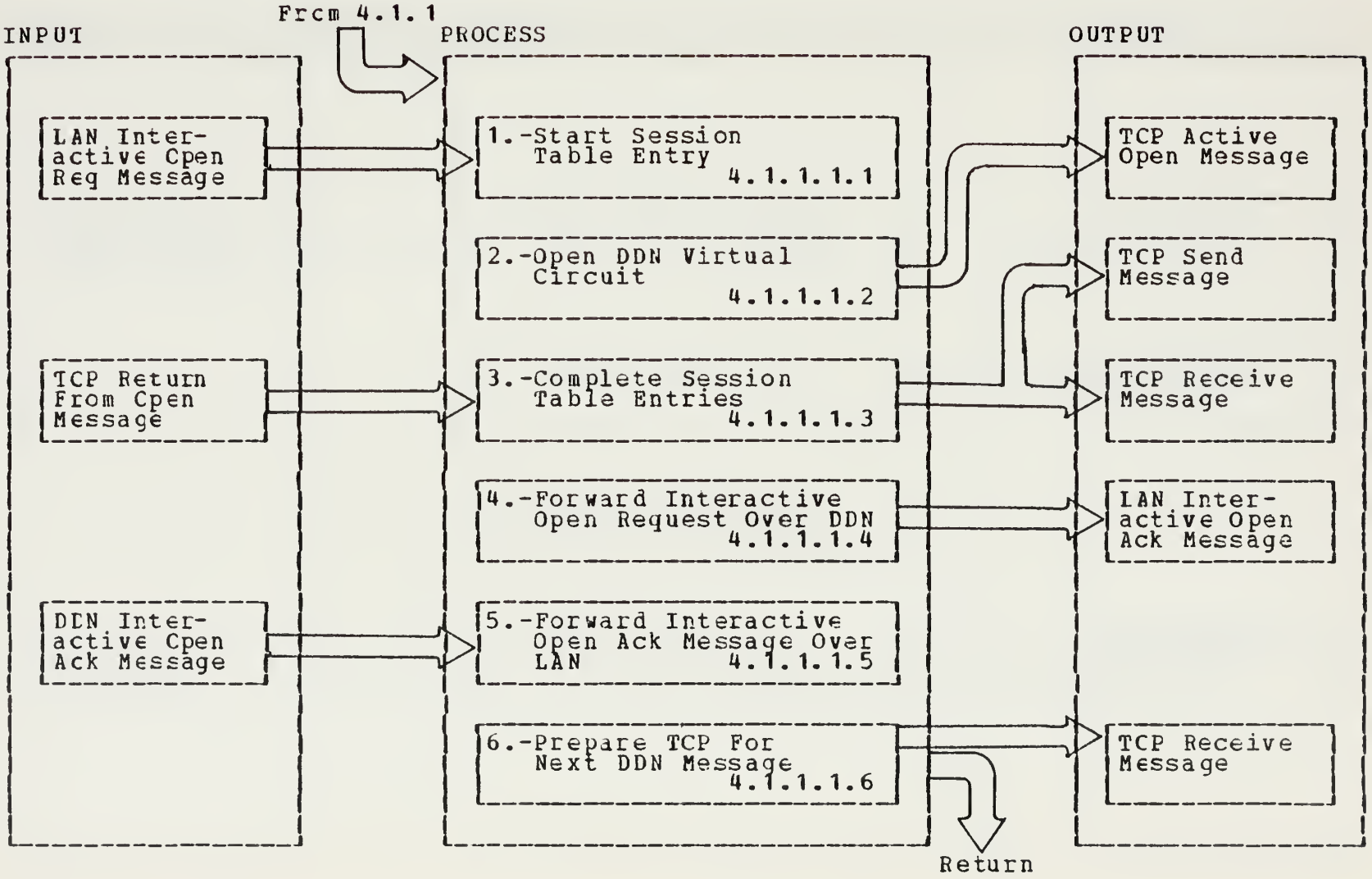
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.1 Name:LAN009 Description:Message Out Service



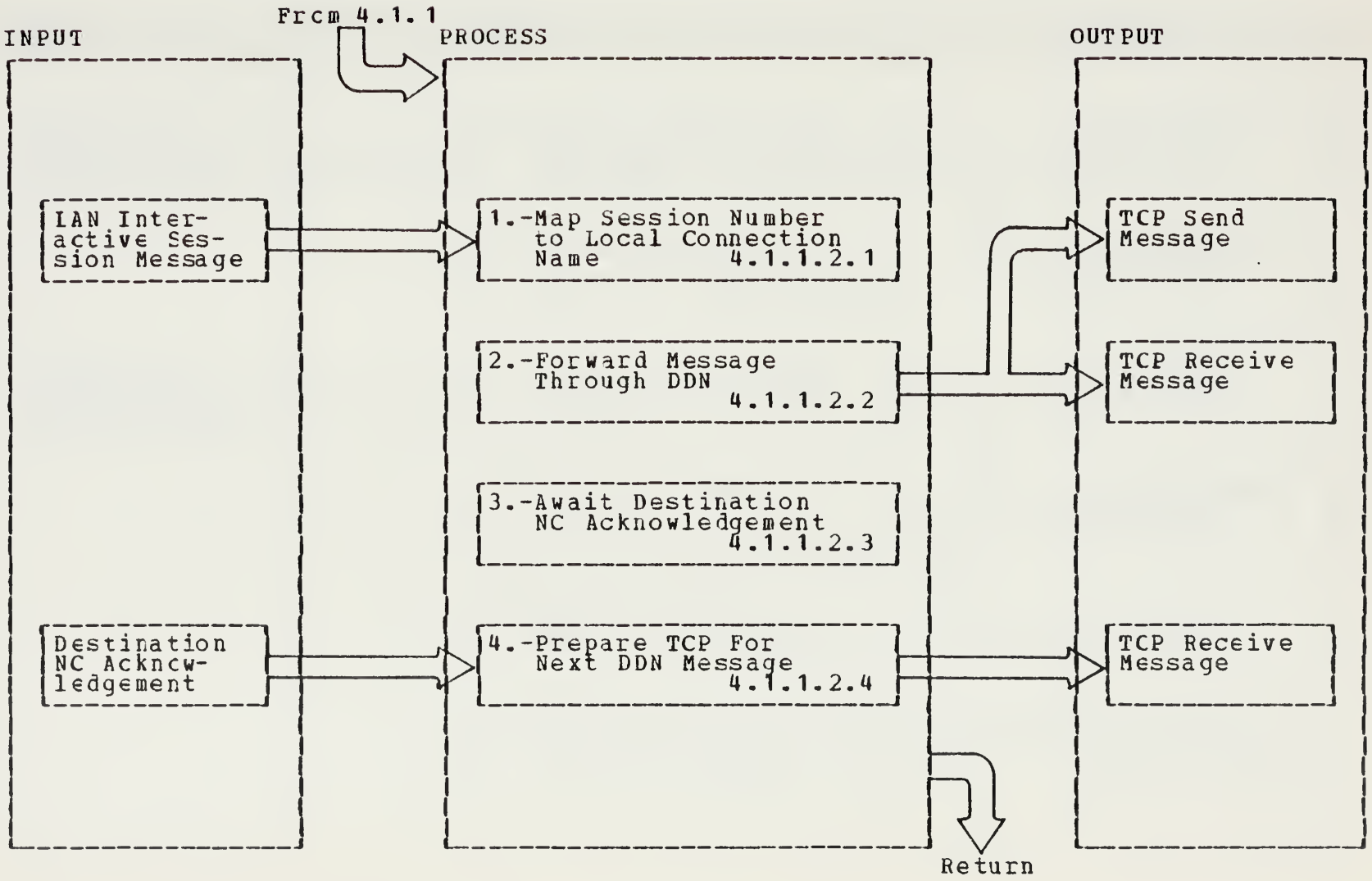
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.1.1 Name:LAN010 Description:Interactive Message Service (Out)



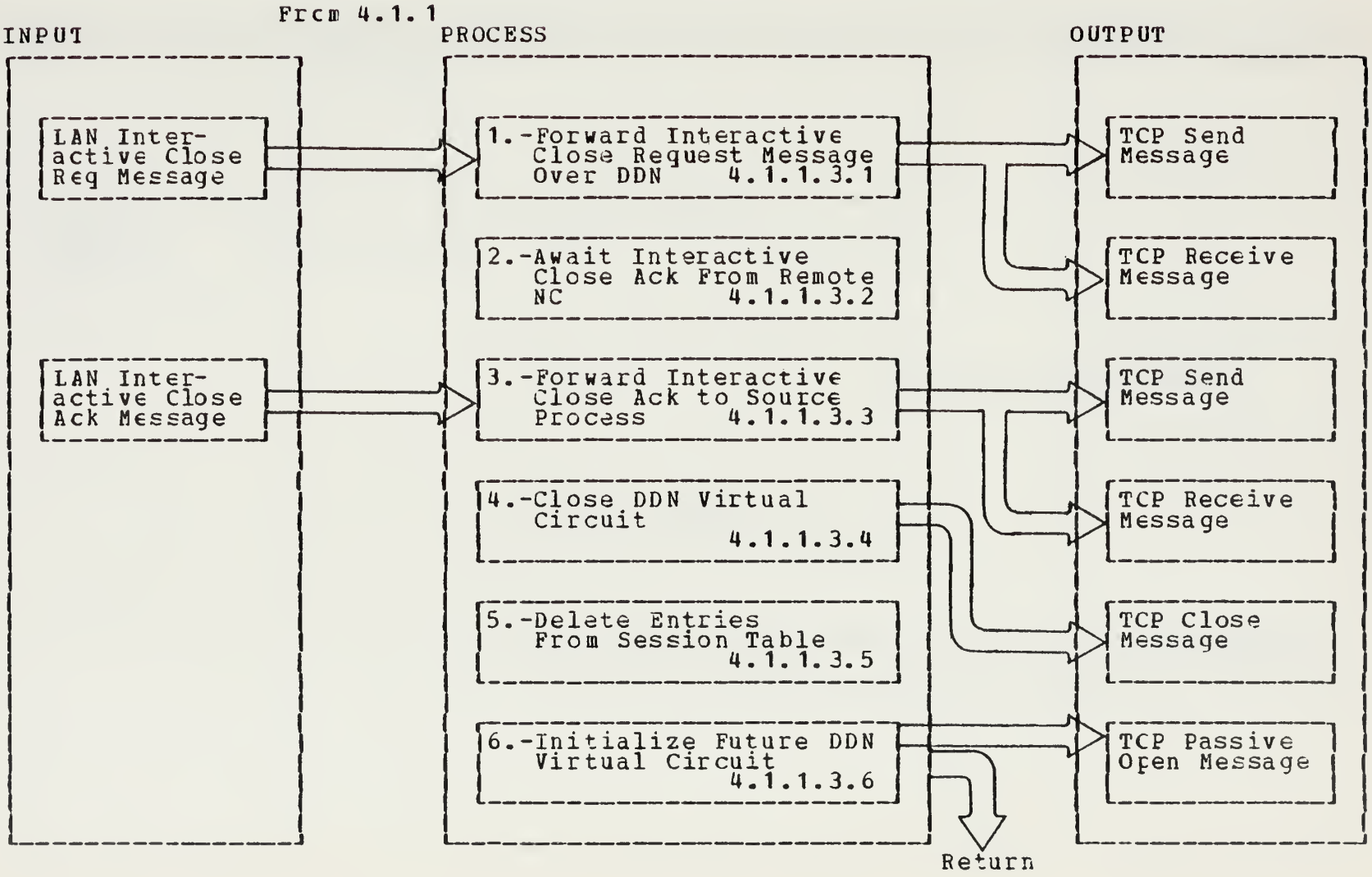
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.1.1.1 Name:LAN011 Description:Open Interactive Session (Out)



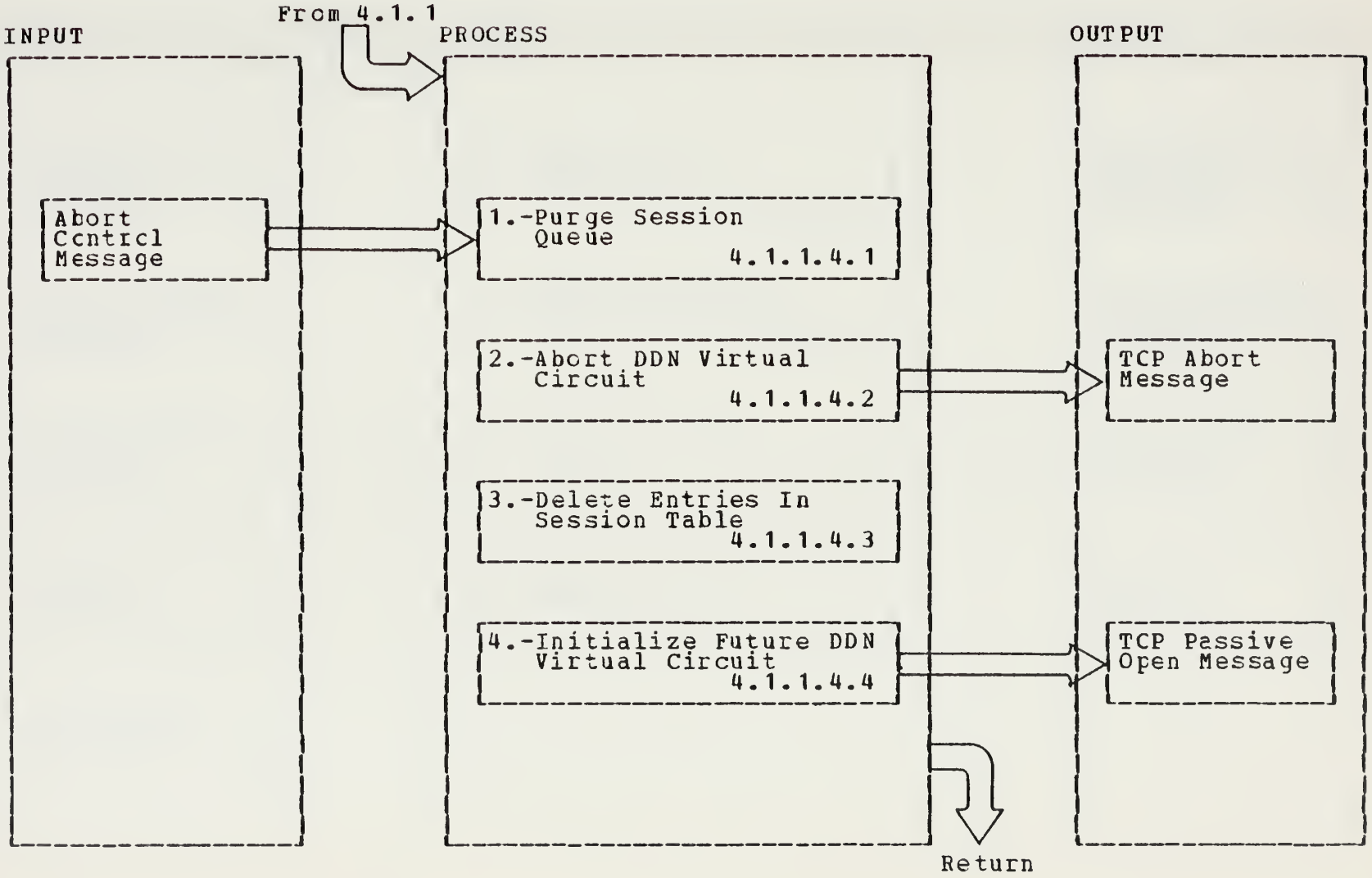
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.1.1.2 Name:LAN012 Description:Interactive Session EstabSvc (Out)



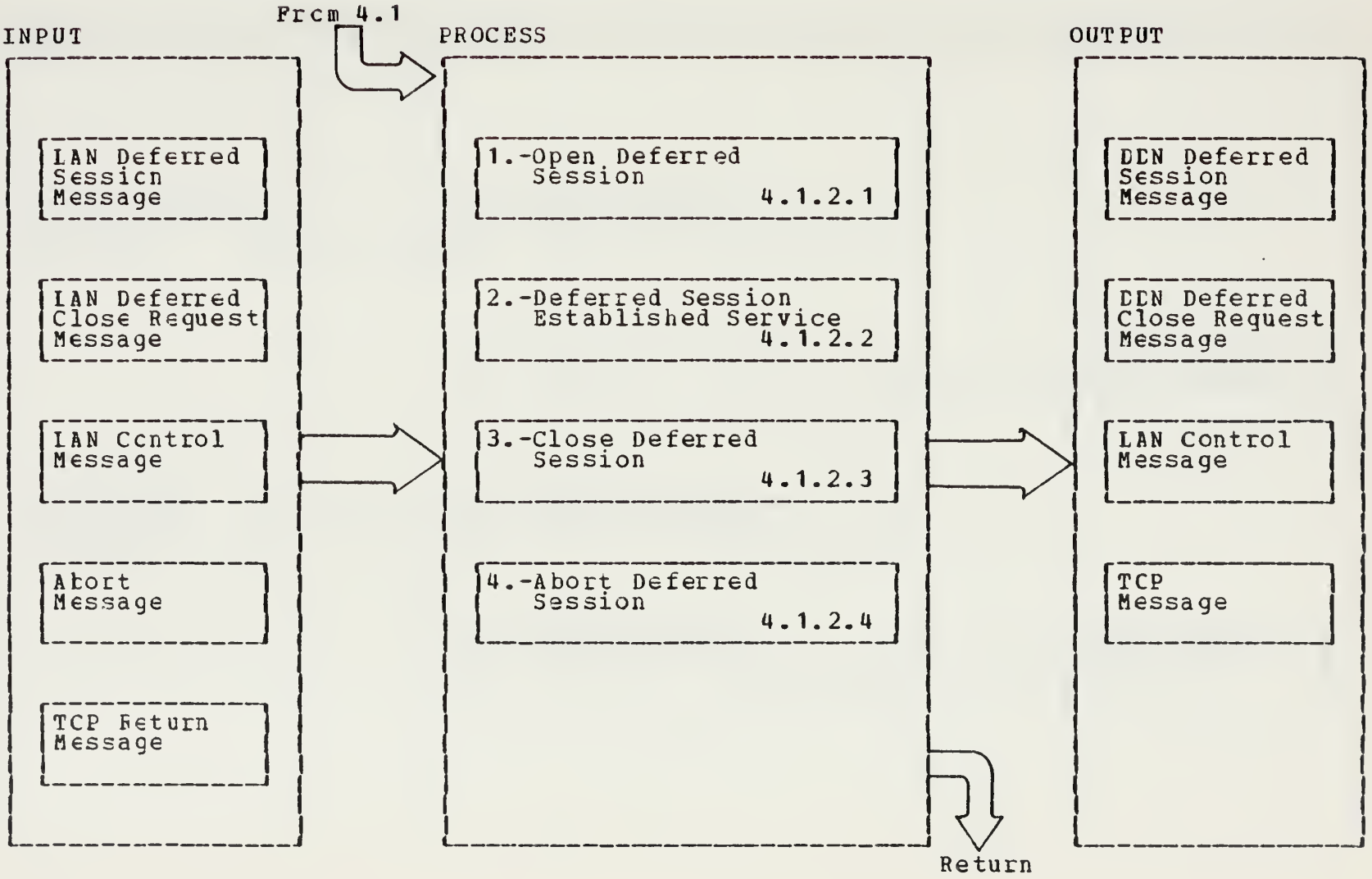
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.1.1.3 Name:LAN013 Description:Close Interactive Session (Out)



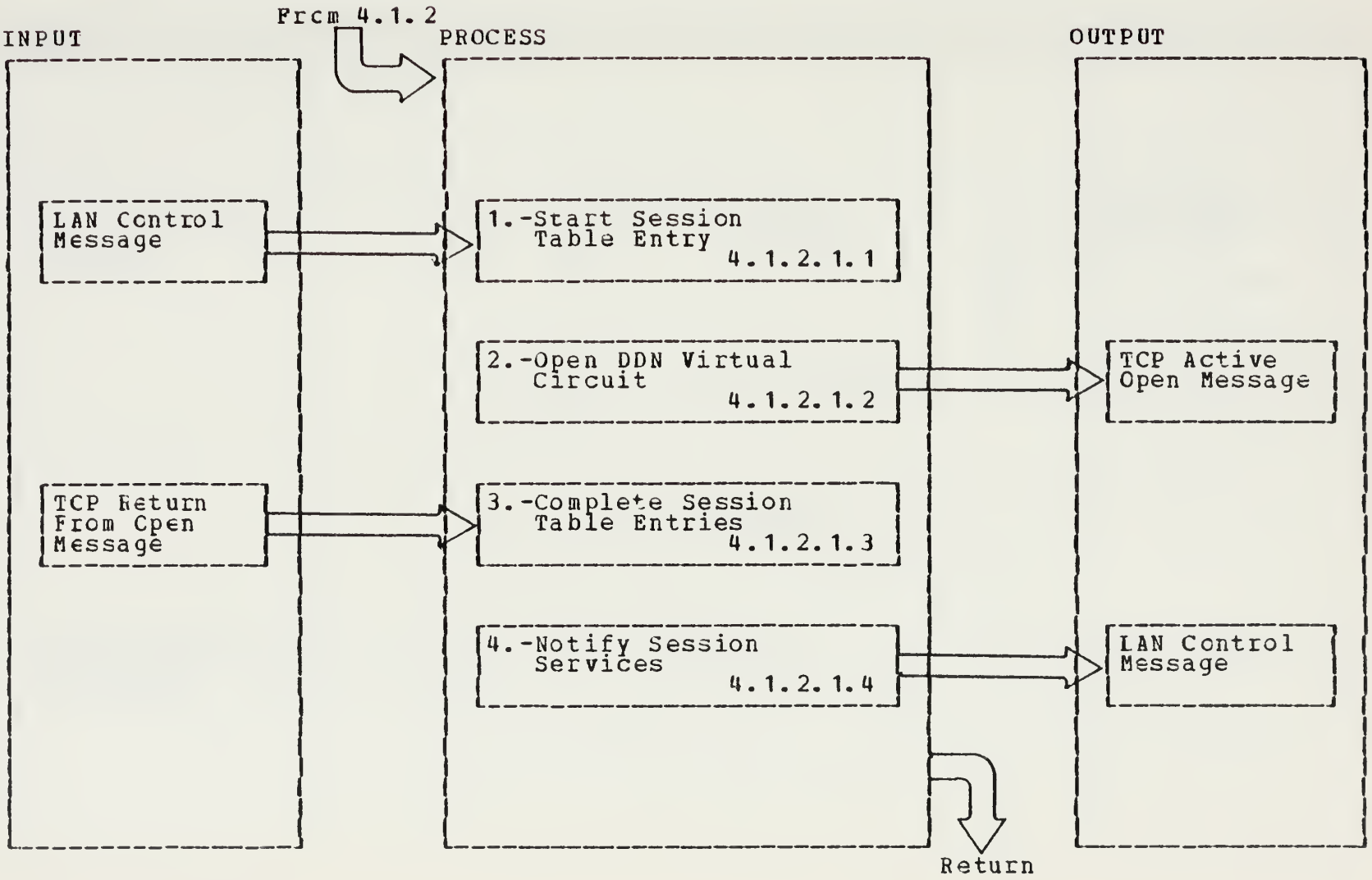
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.1.1.4 Name:LAN014 Description:Abort Interactive Session (Out)



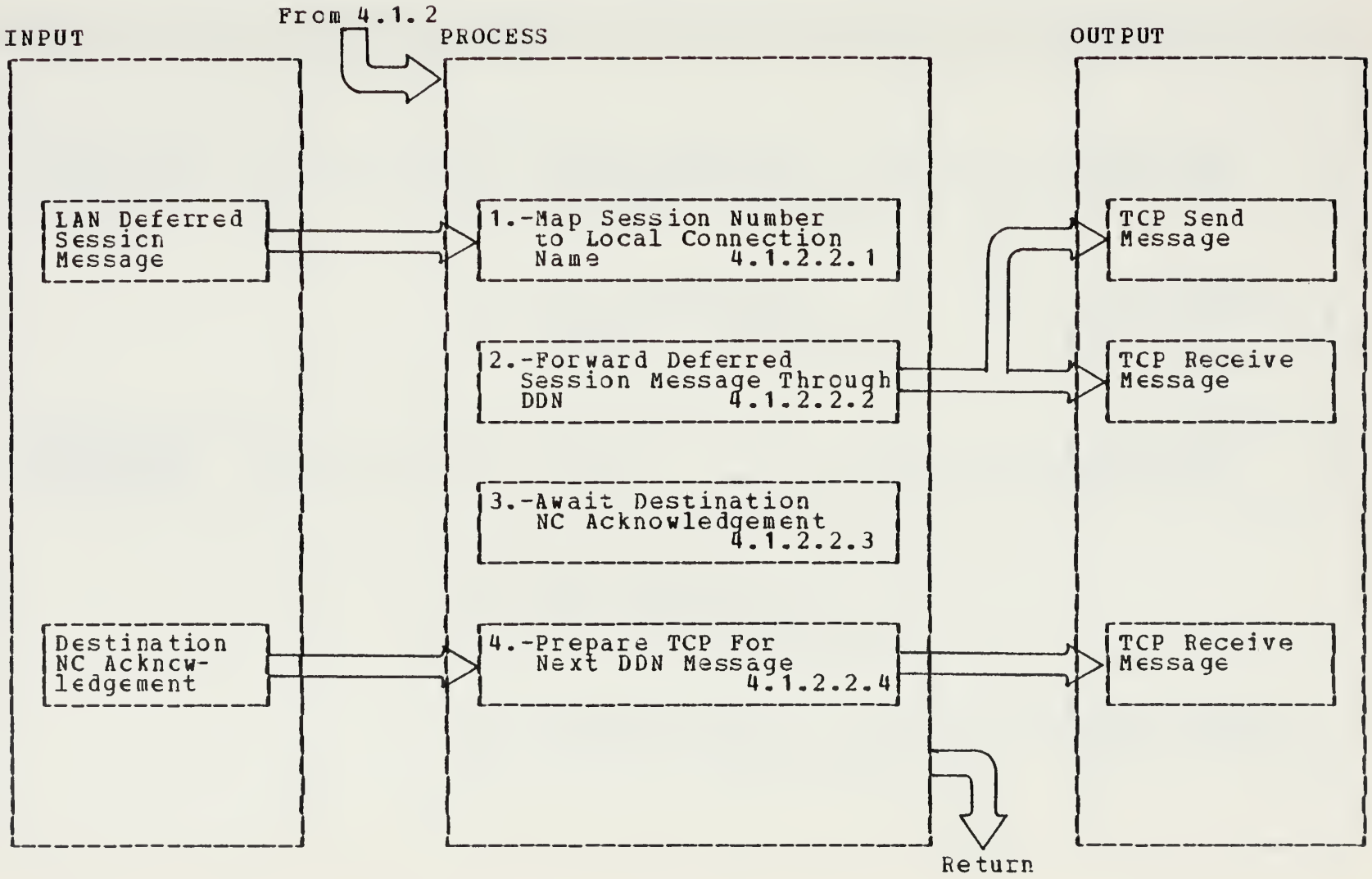
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.1.2 Name:LAN015 Description:Deferred Message Service (Out)



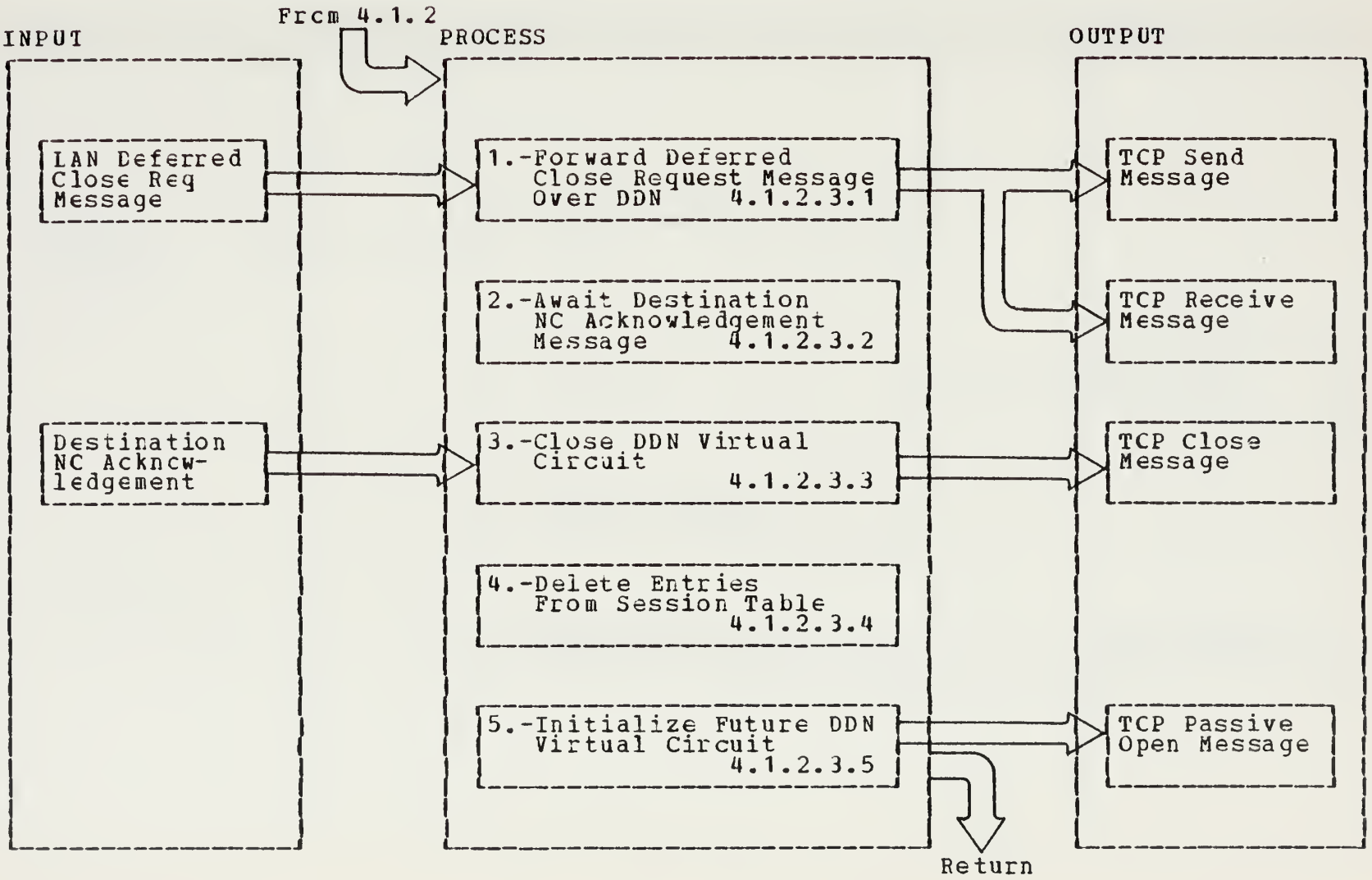
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.1.2.1 Name:LAN016 Description:Open Deferred Session (Cut)



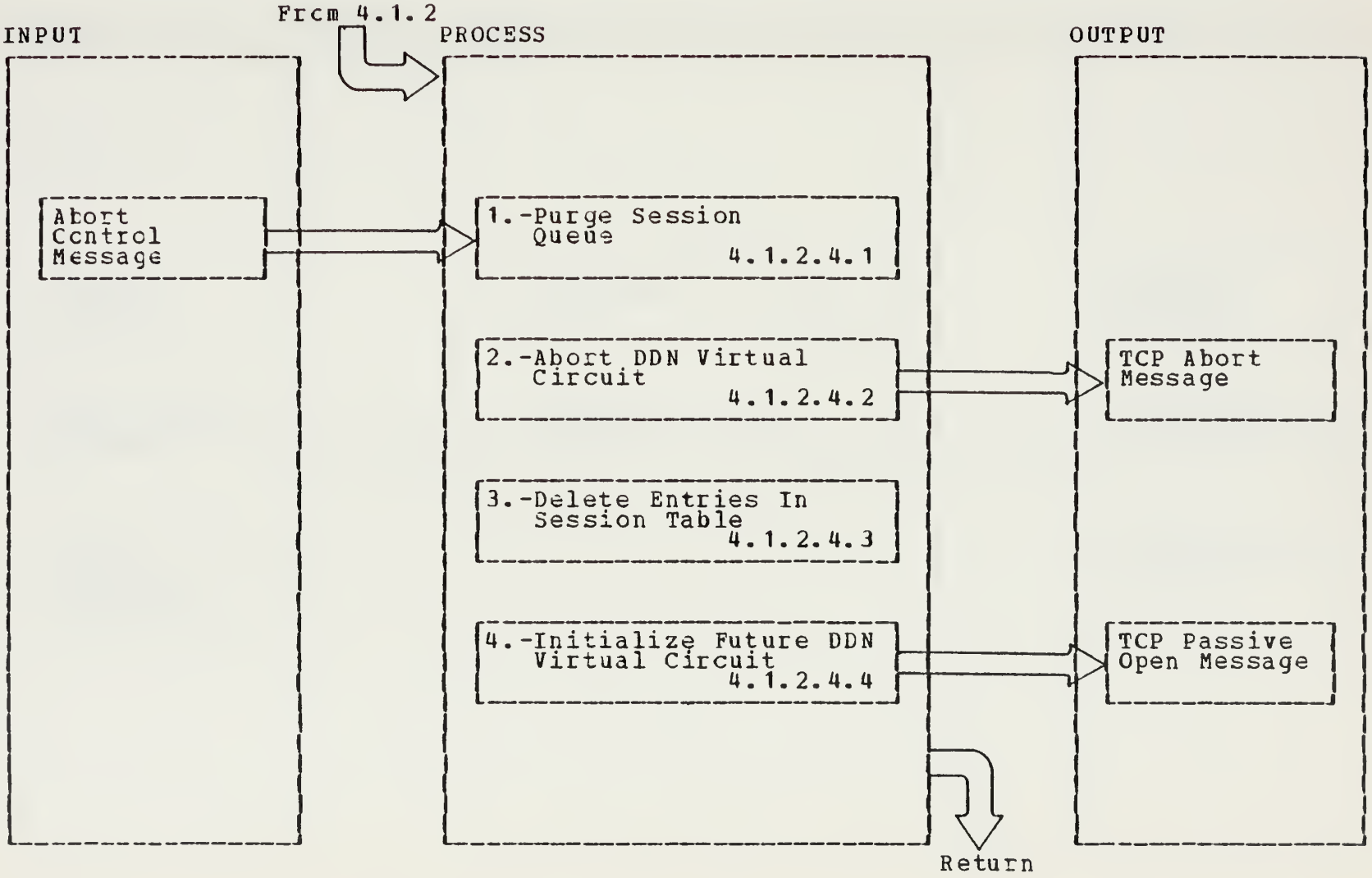
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.1.2.2 Name:LAN017 Description:Deferred Session Estab Svc (Out)



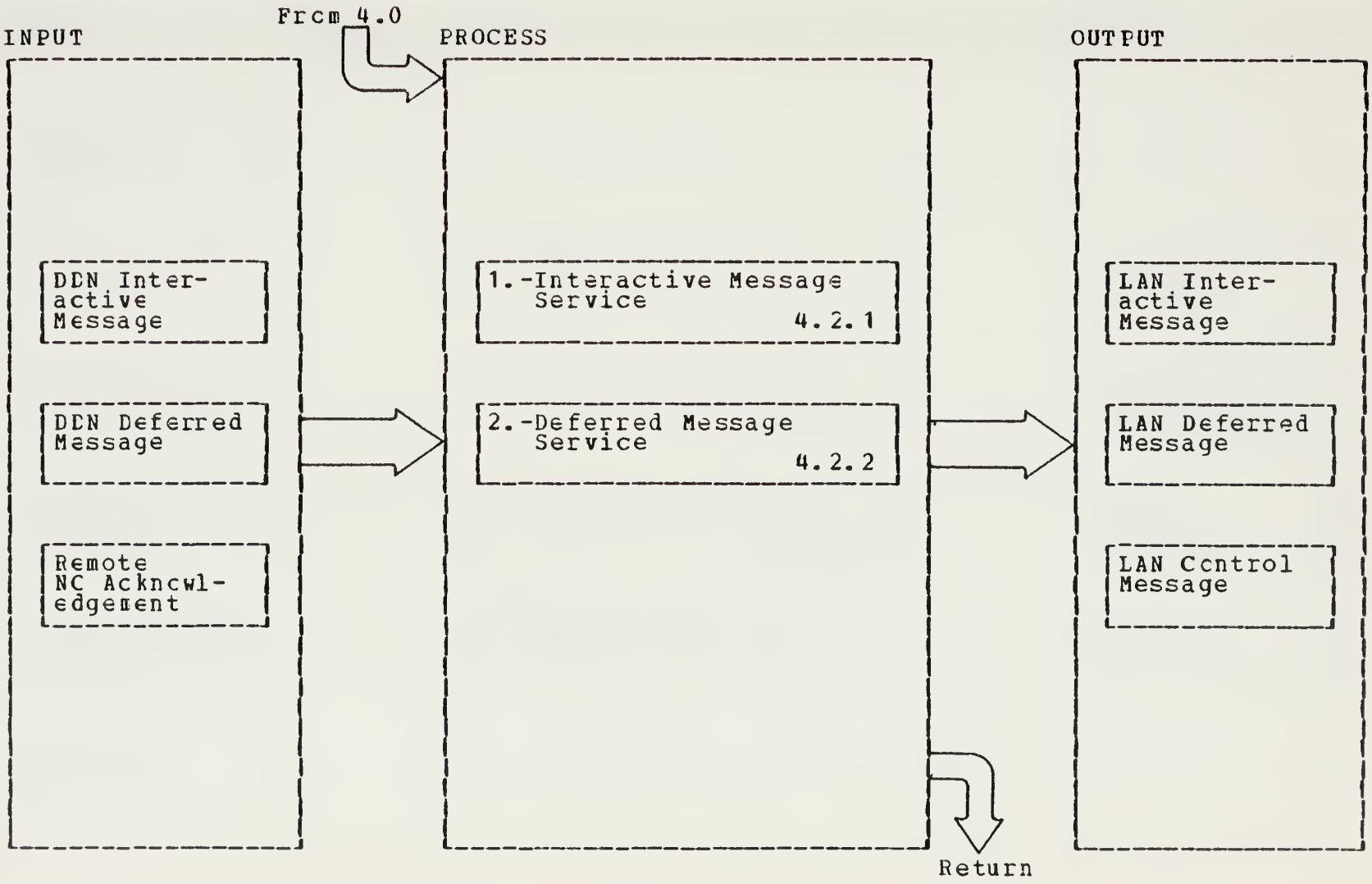
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.1.2.3 Name:LAN018 Description:Close Deferred Session (Out)



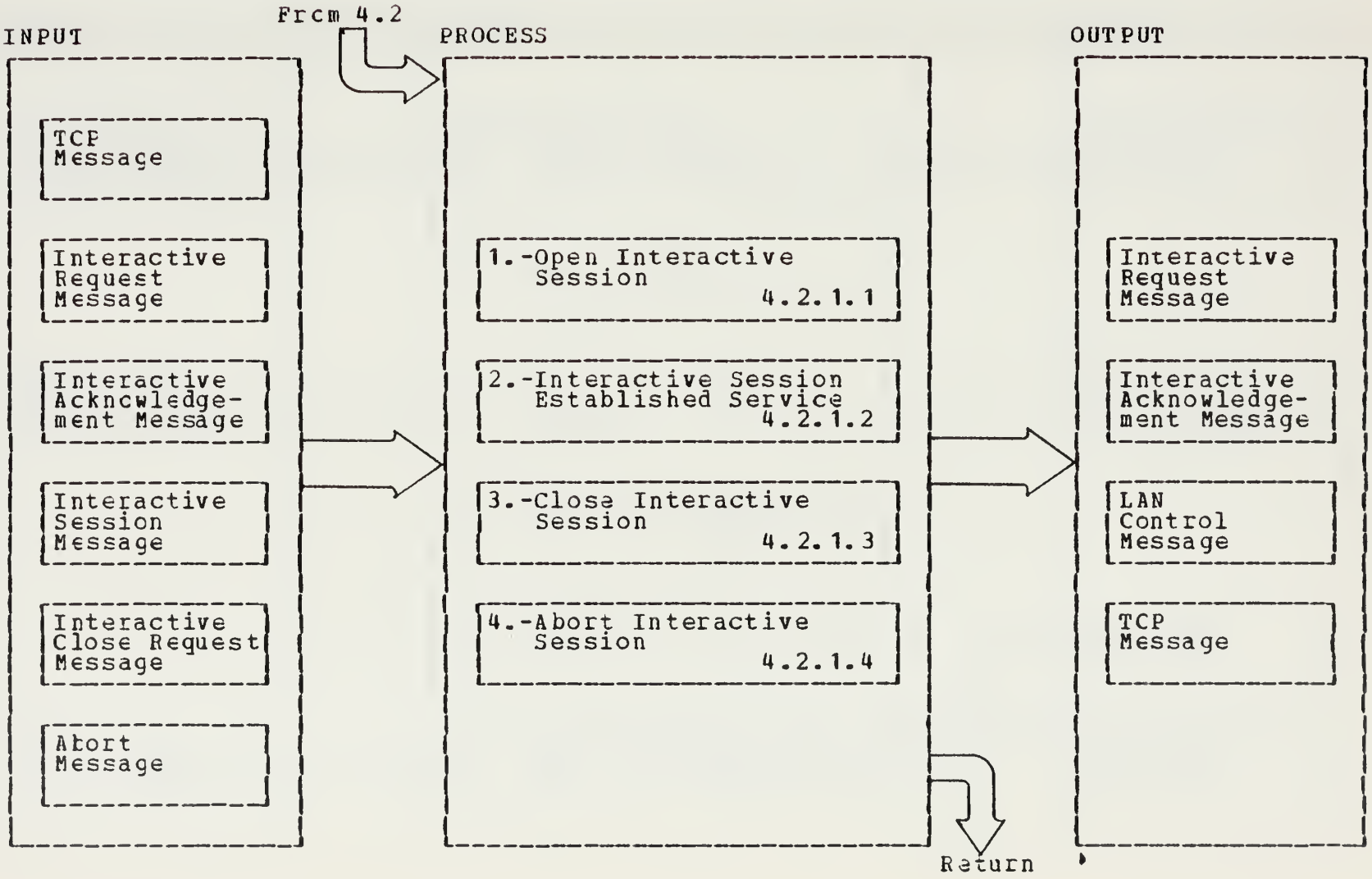
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.1.2.4 Name:LAN019 Description:Abort Deferred Session (Out)



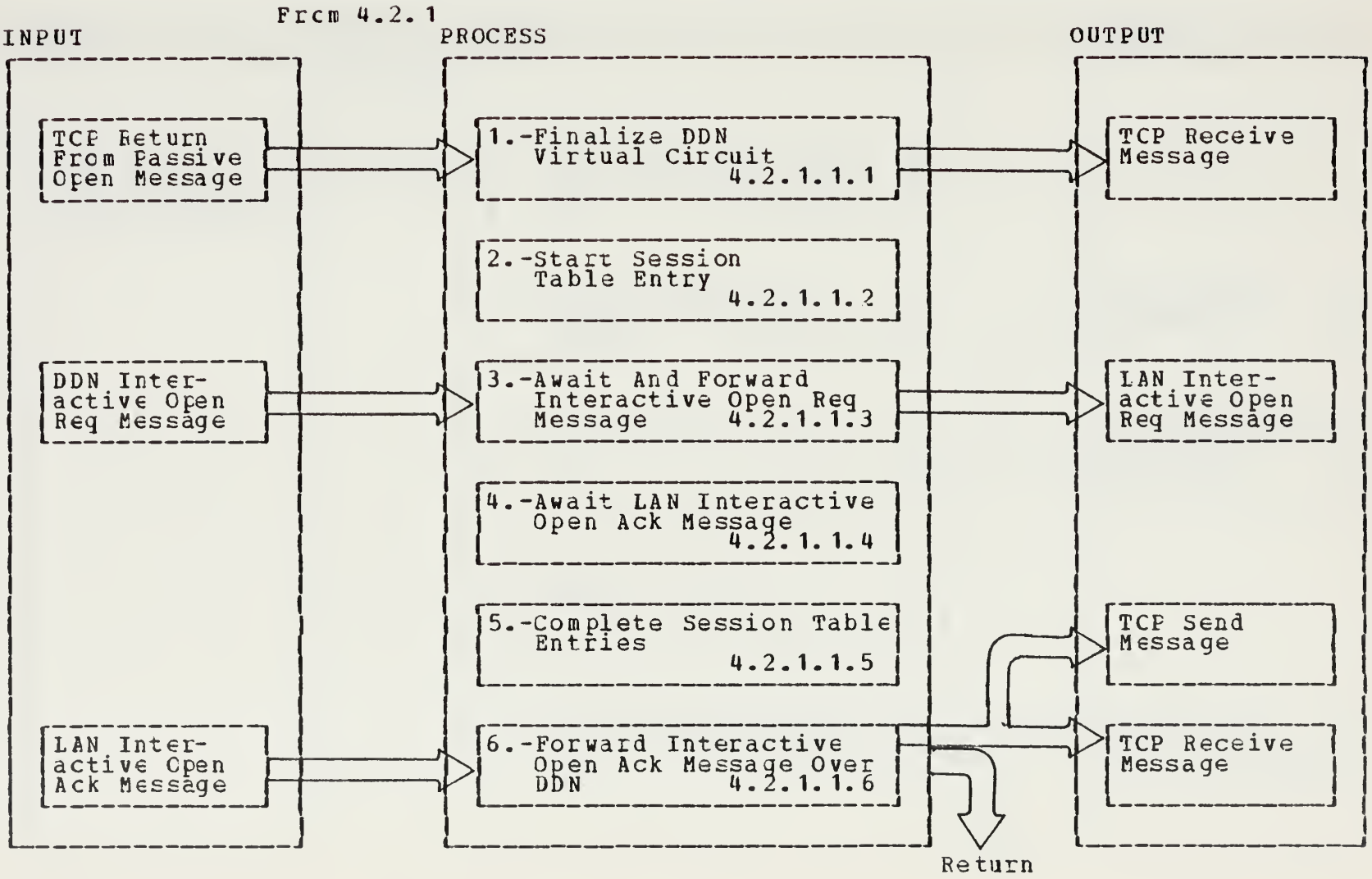
Authcr:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.2 Name:LAN020 Description:Message In Service



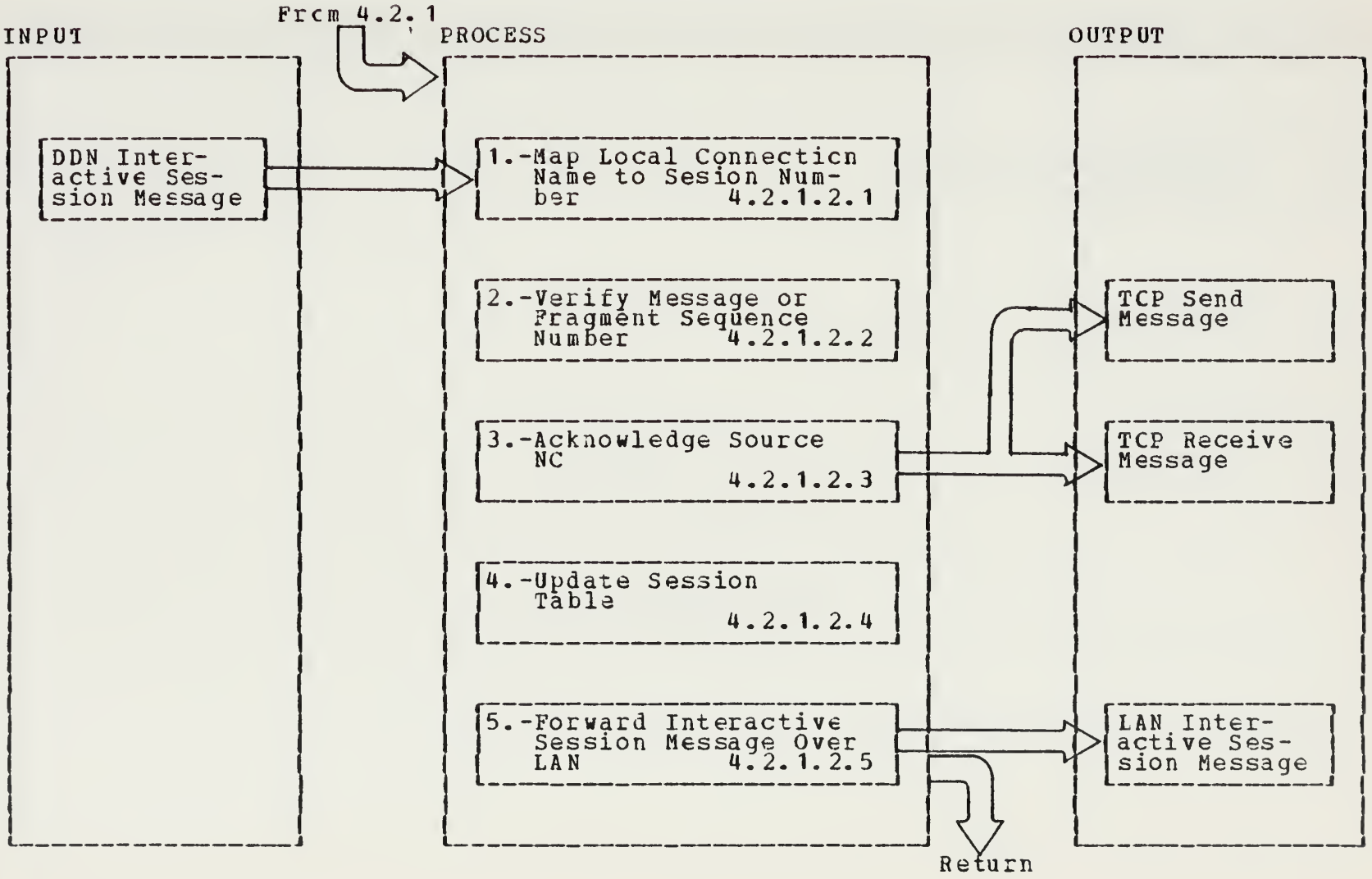
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.2.1 Name:LAN021 Description:Interactive Message Service (In)



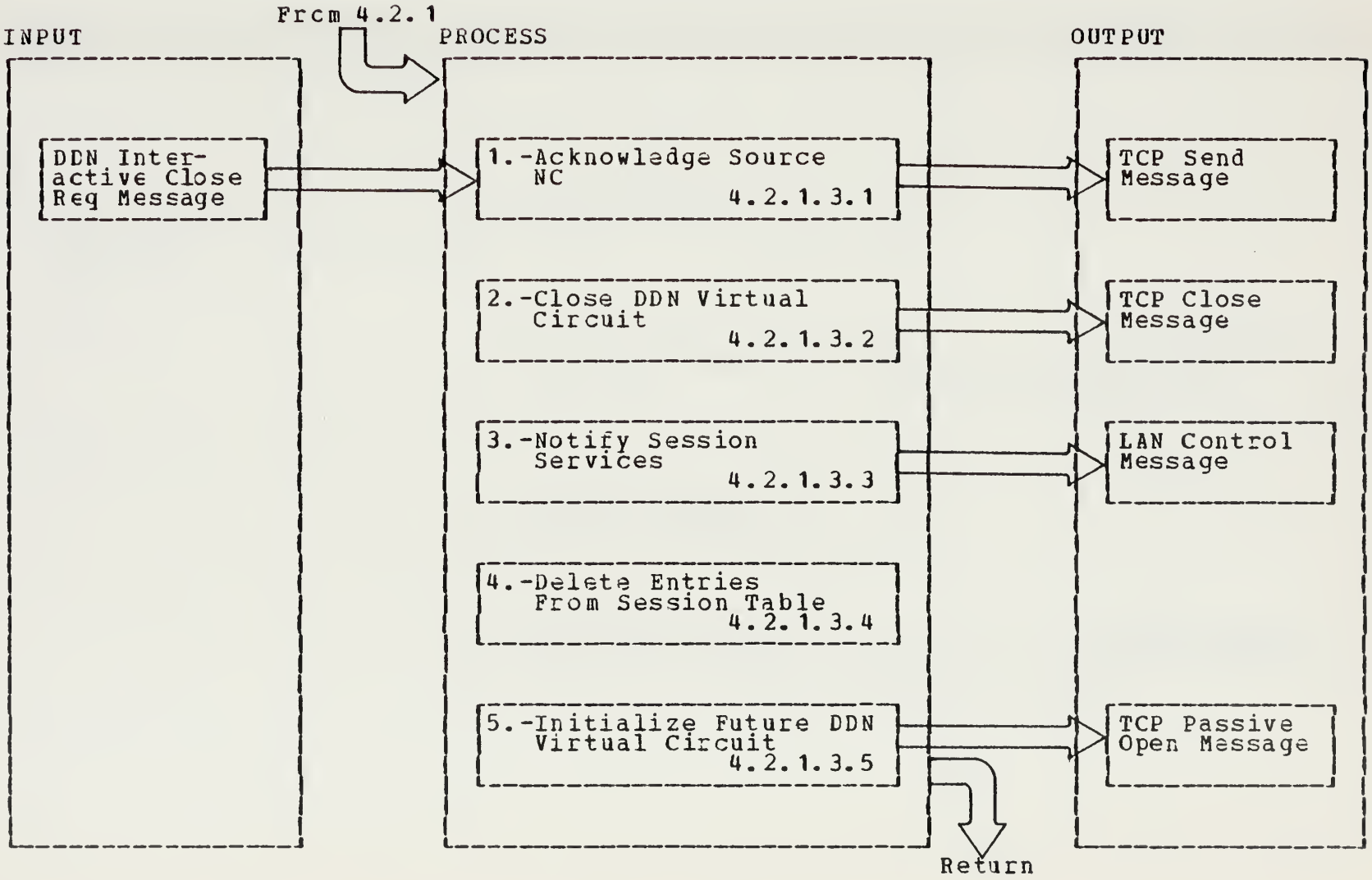
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.2.1.1 Name:LAN022 Description: Open Interactive Session (In)



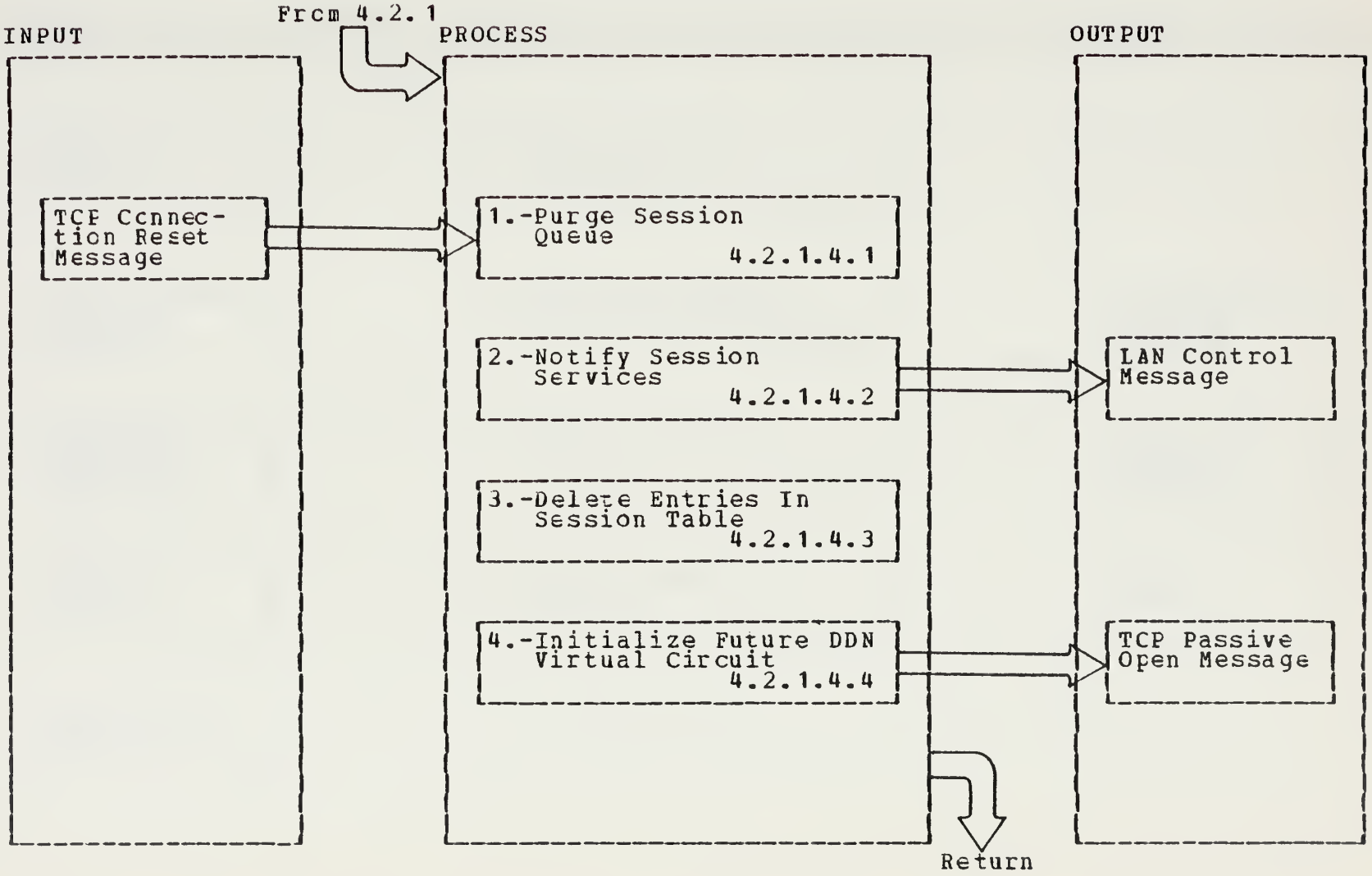
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.2.1.2 Name:LAN023 Description:Interactive Session EstabSvc(In)



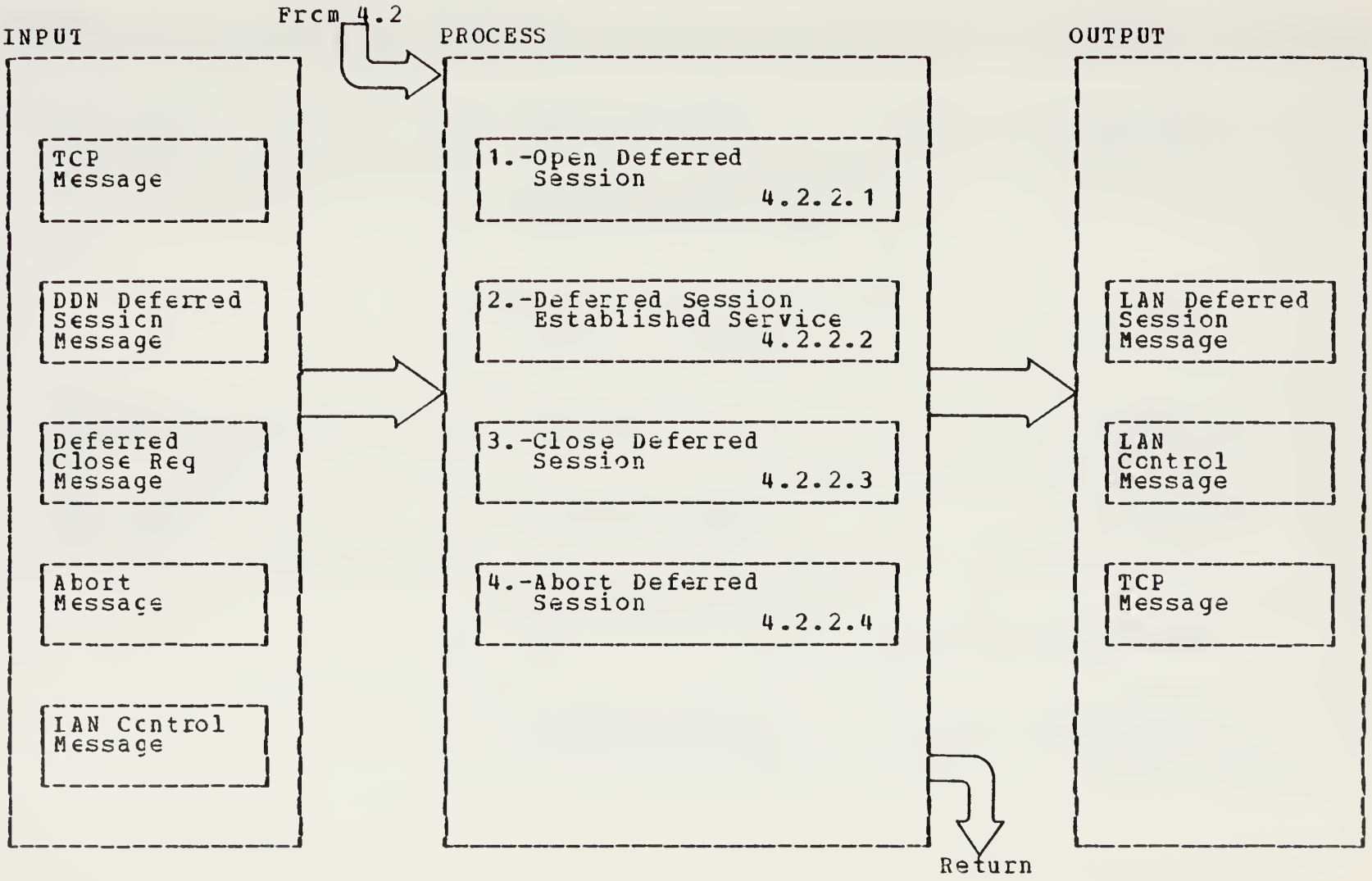
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.2.1.3 Name:LAN024 Description:Close Interactive Session (In)



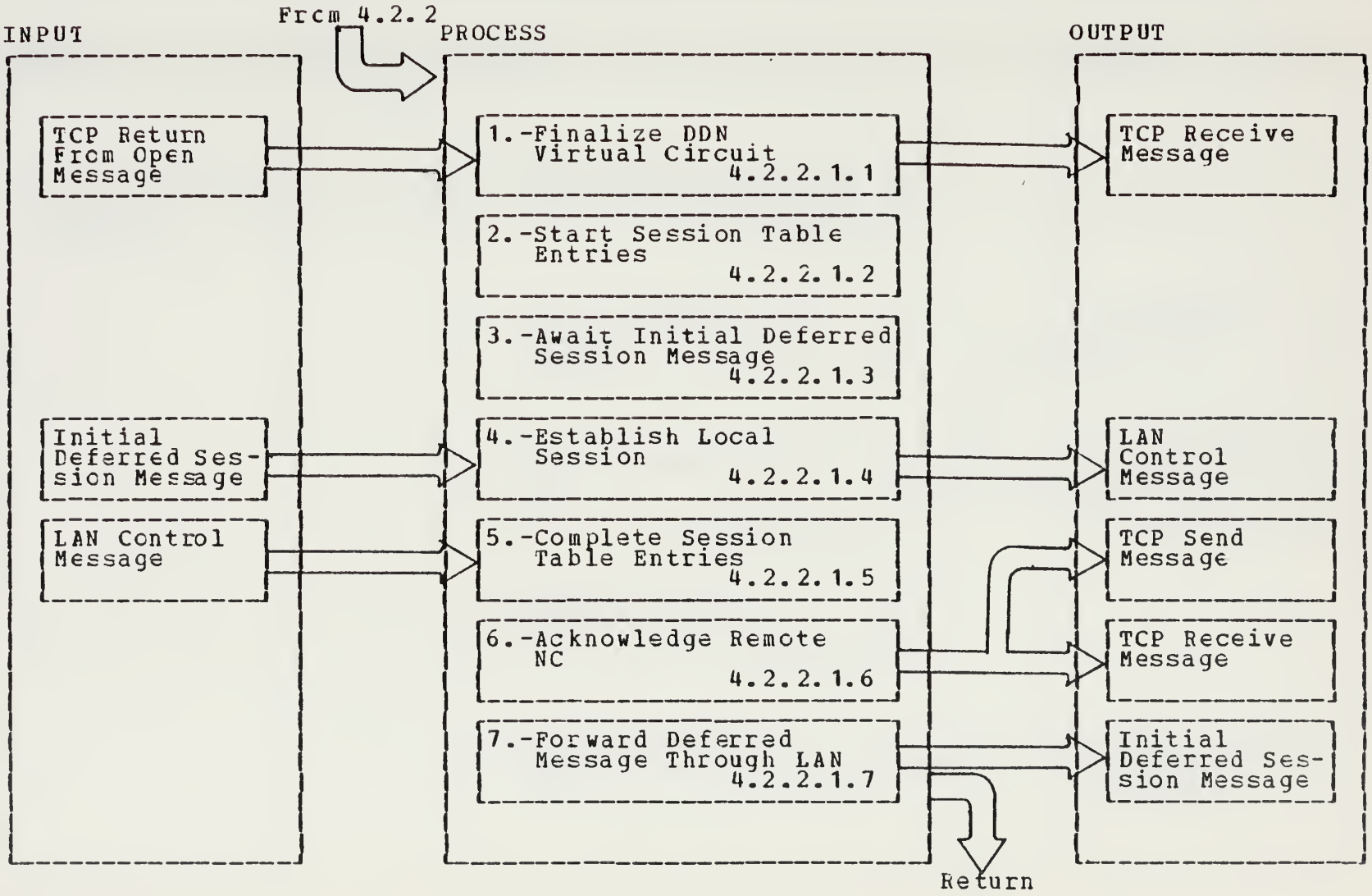
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.2.1.4 Name:LAN025 Description:Abort Interactive Session (In)



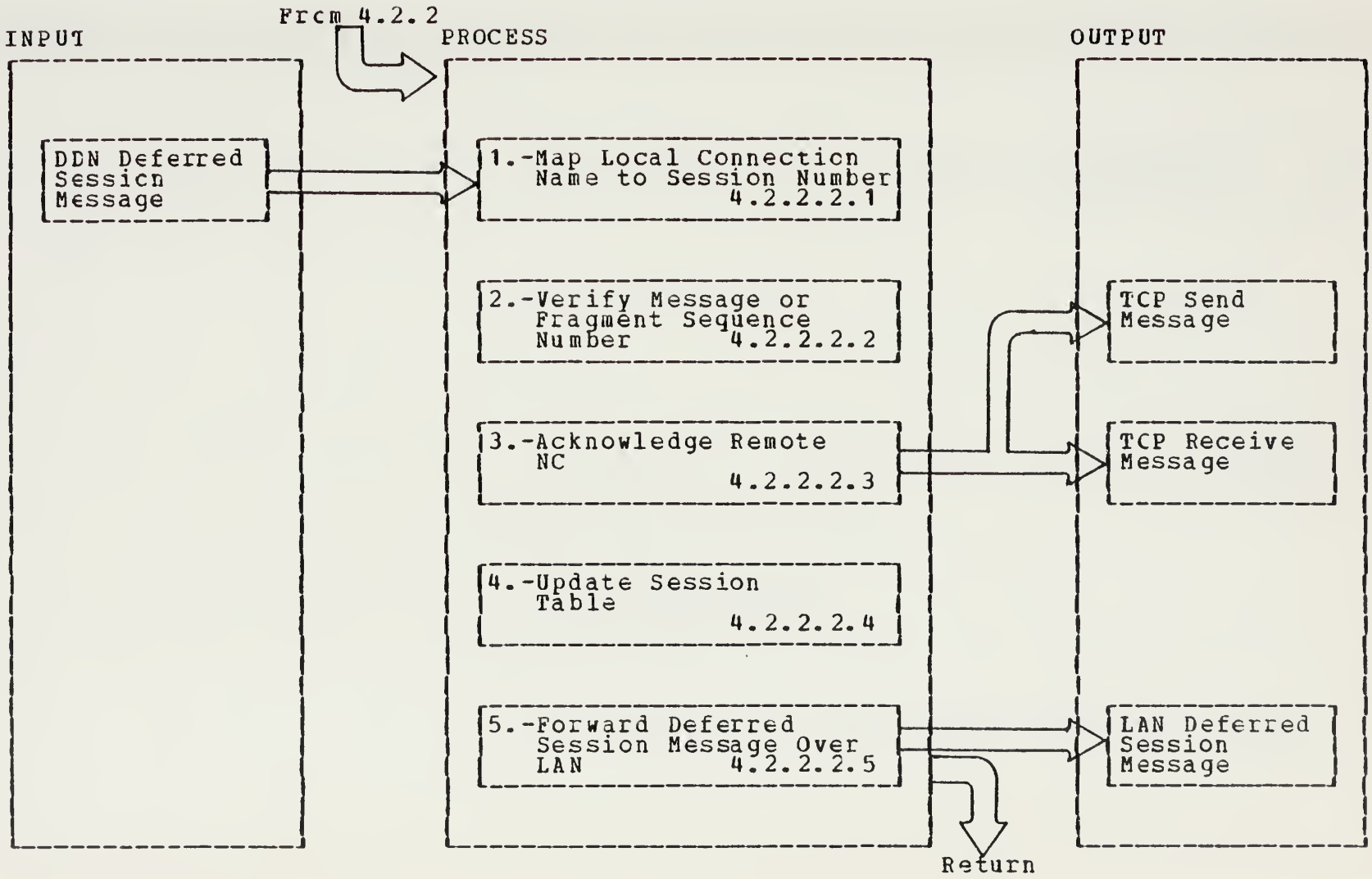
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.2.2 Name:LAN026 Description:Deferred Message Service (In)



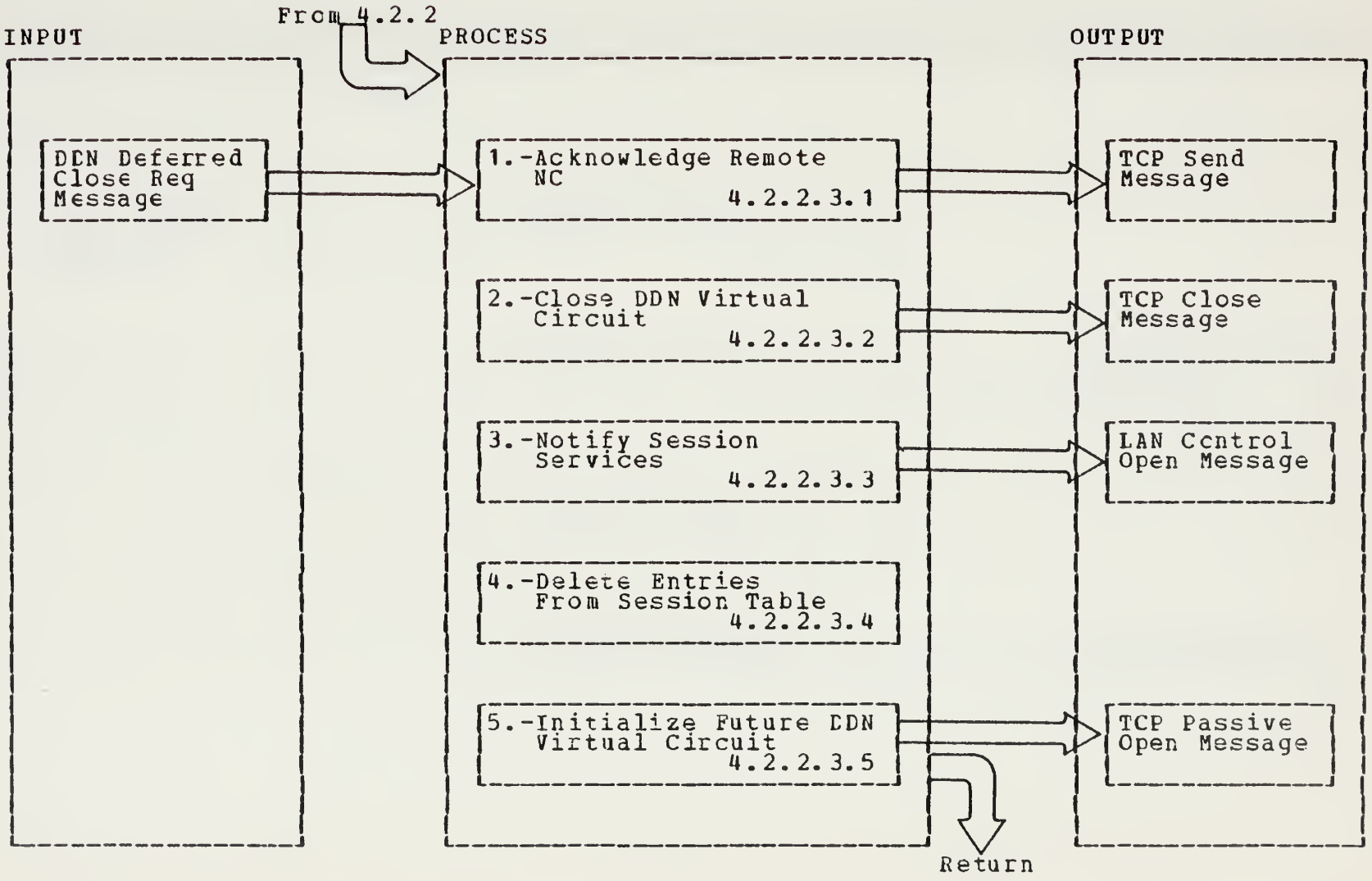
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.2.2.1 Name:LAN027 Description: Open Deferred Session (In)



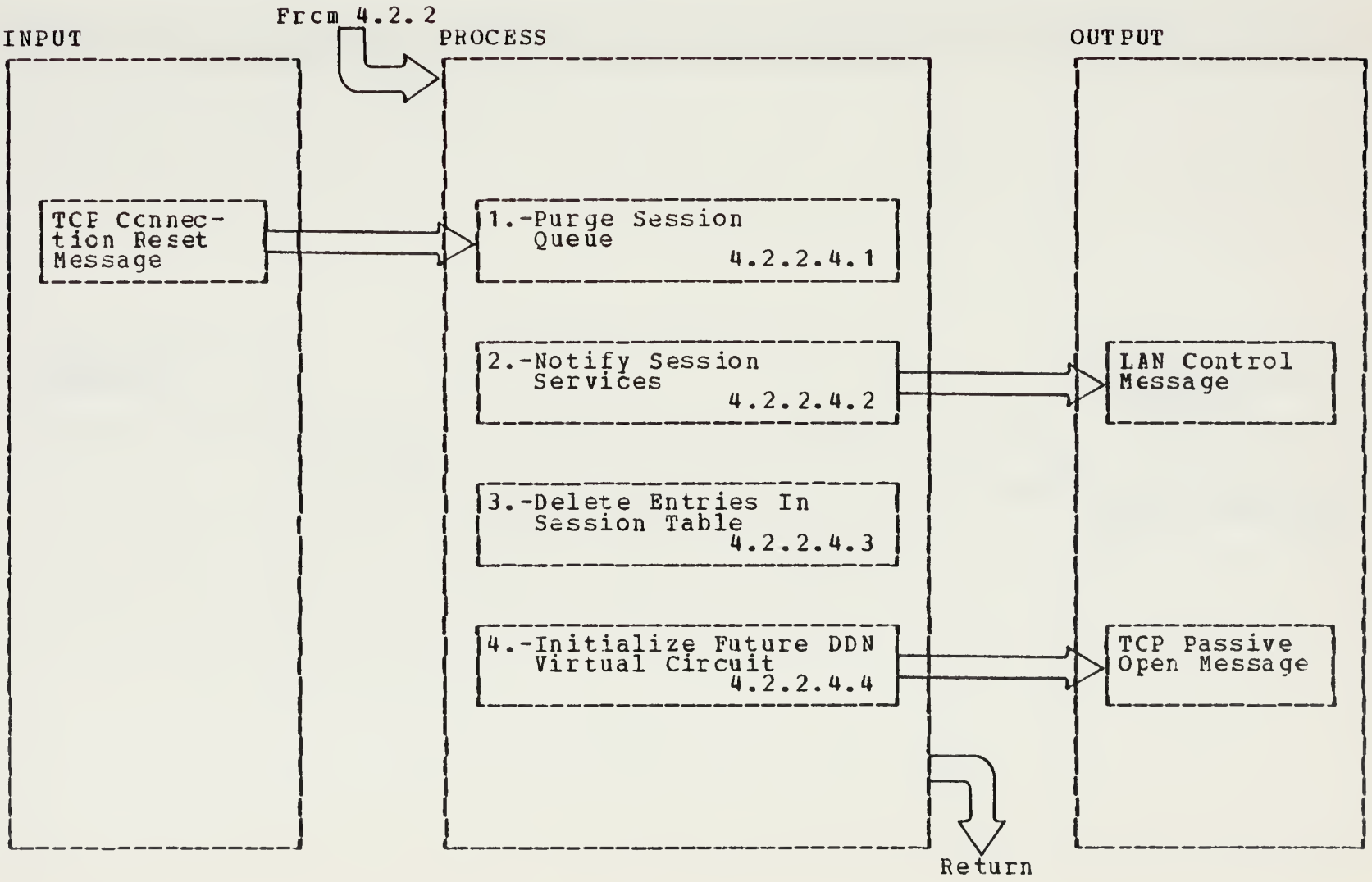
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.2.2.2 Name:LAN028 Description:Deferred Session Estab Svc (In)



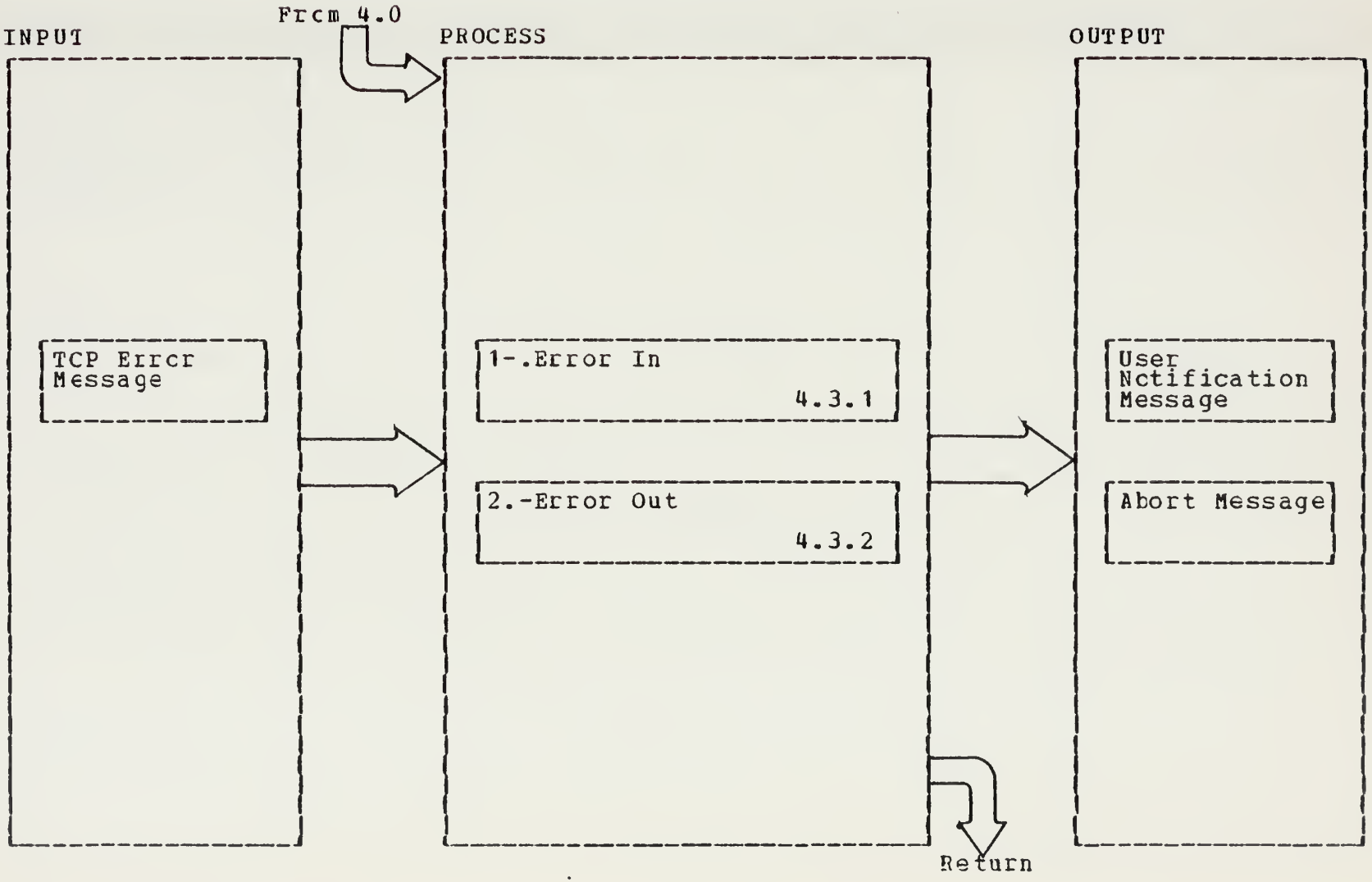
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.2.2.3 Name:LAN029 Description:Close Deferred Session (In)



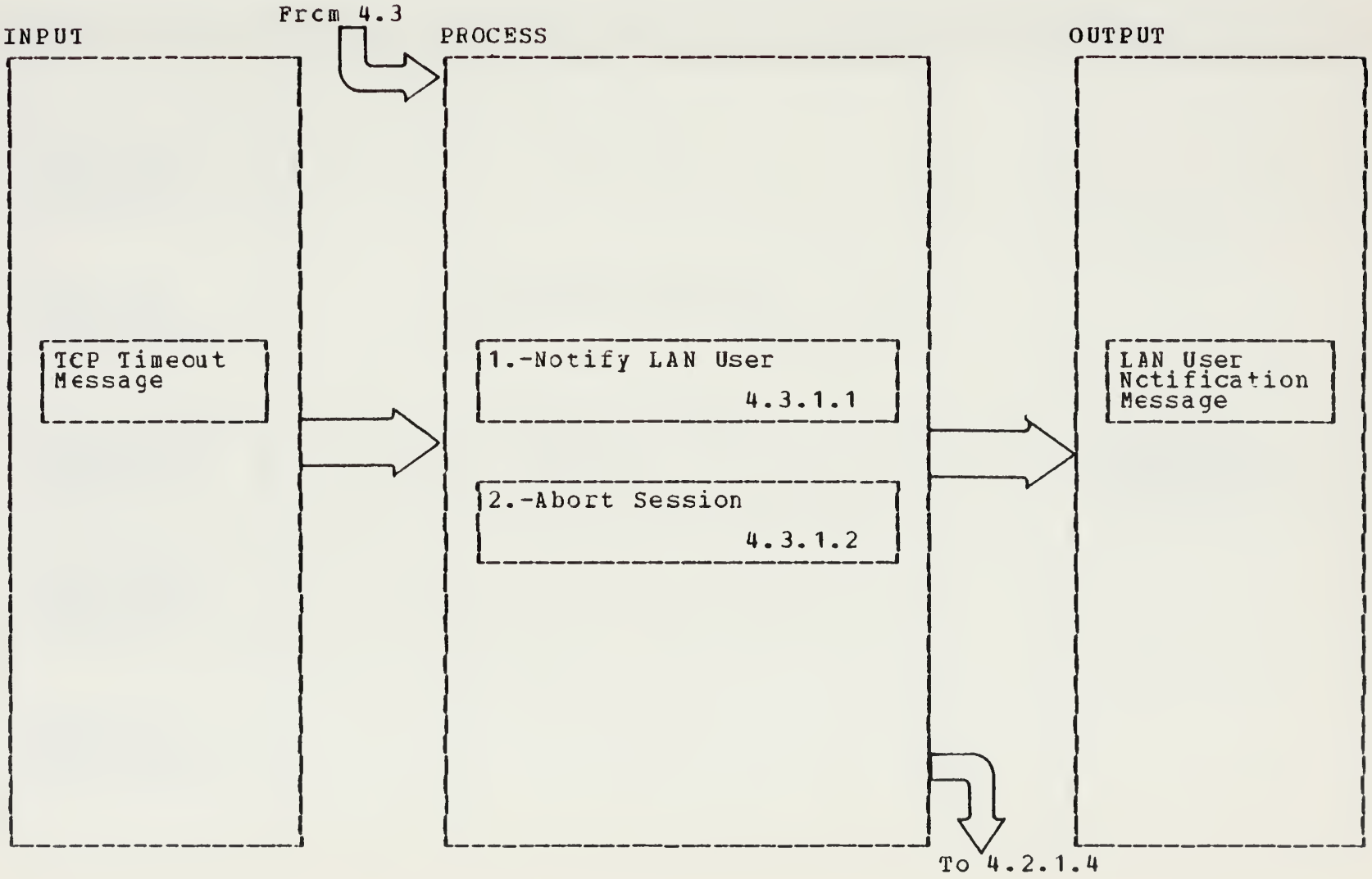
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.2.2.4 Name:LAN030 Description:Abort Deferred Session (In)



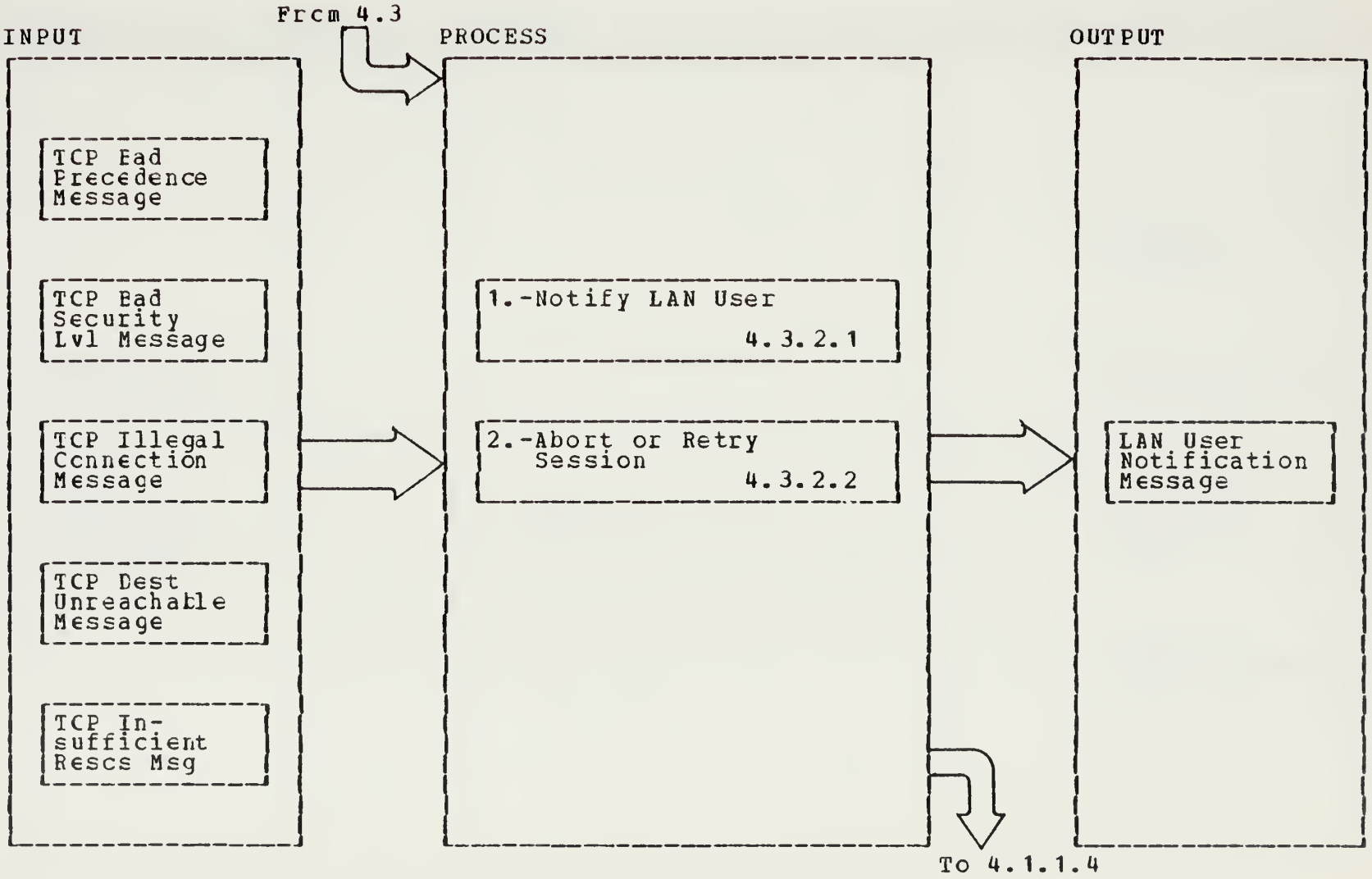
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.3 Name:LAN031 Description:Error Handler



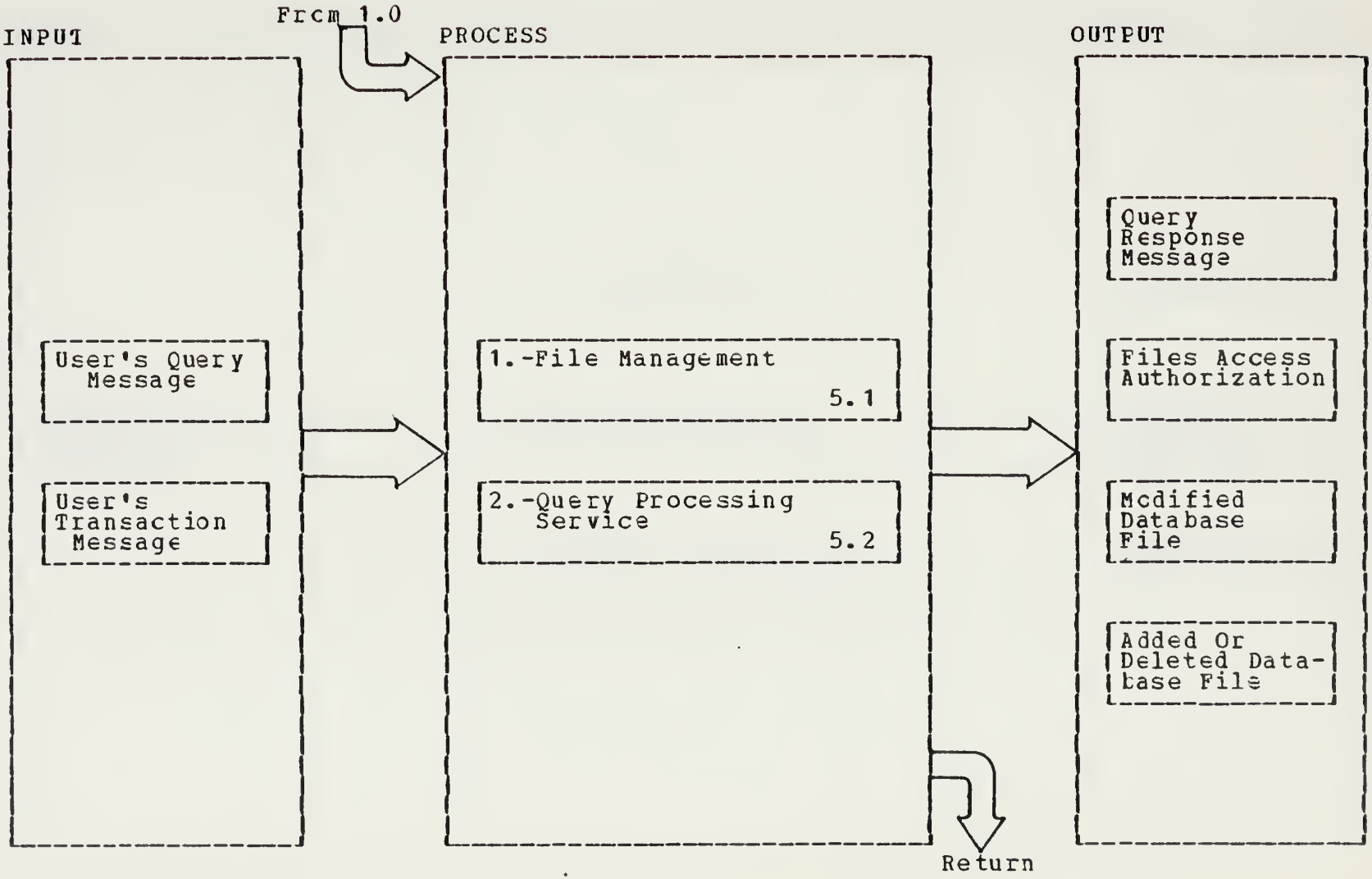
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.3.1 Name:LAN032 Description:Error In



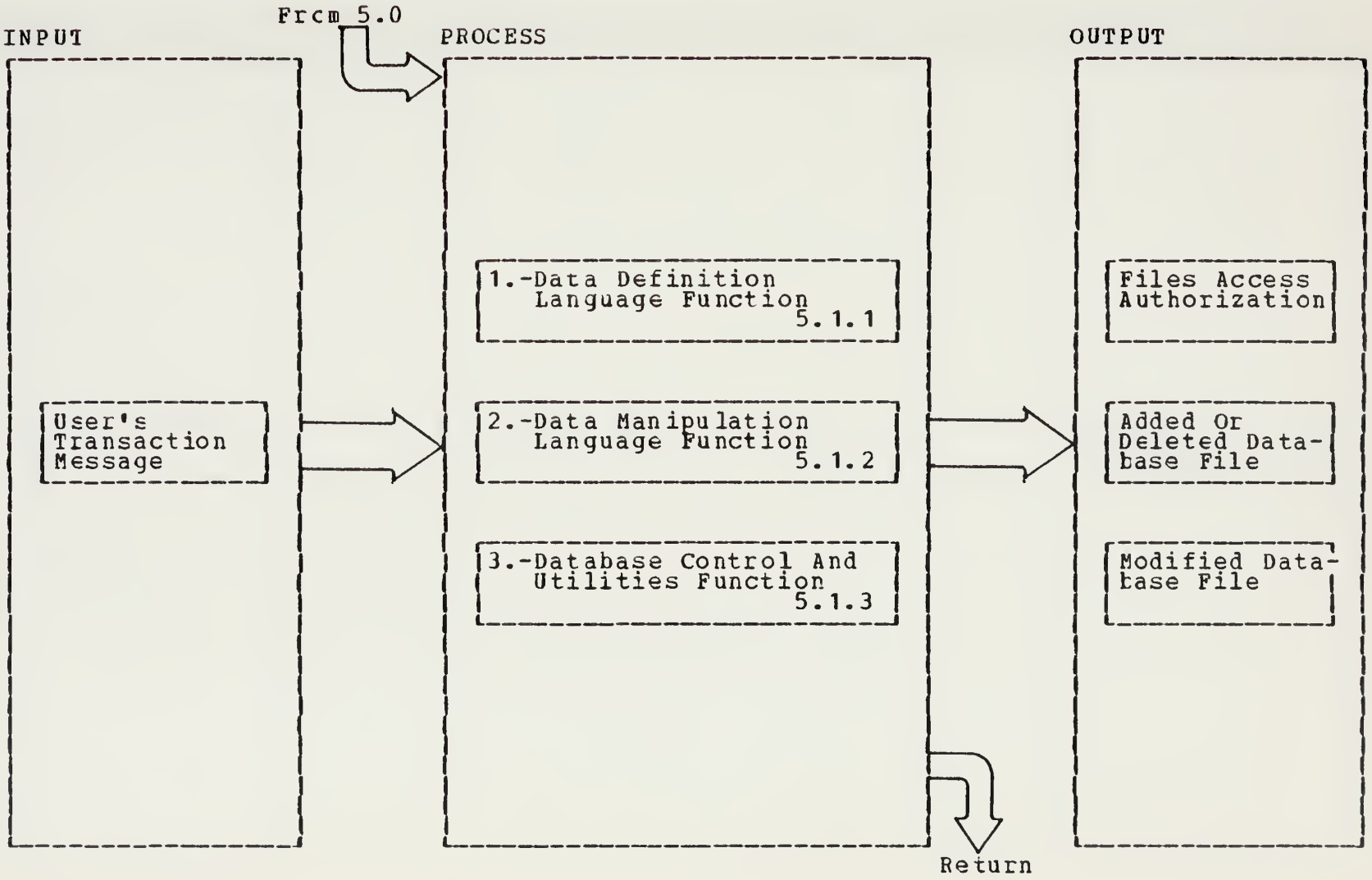
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.3.2 Name:LAN033 Description:Error Out



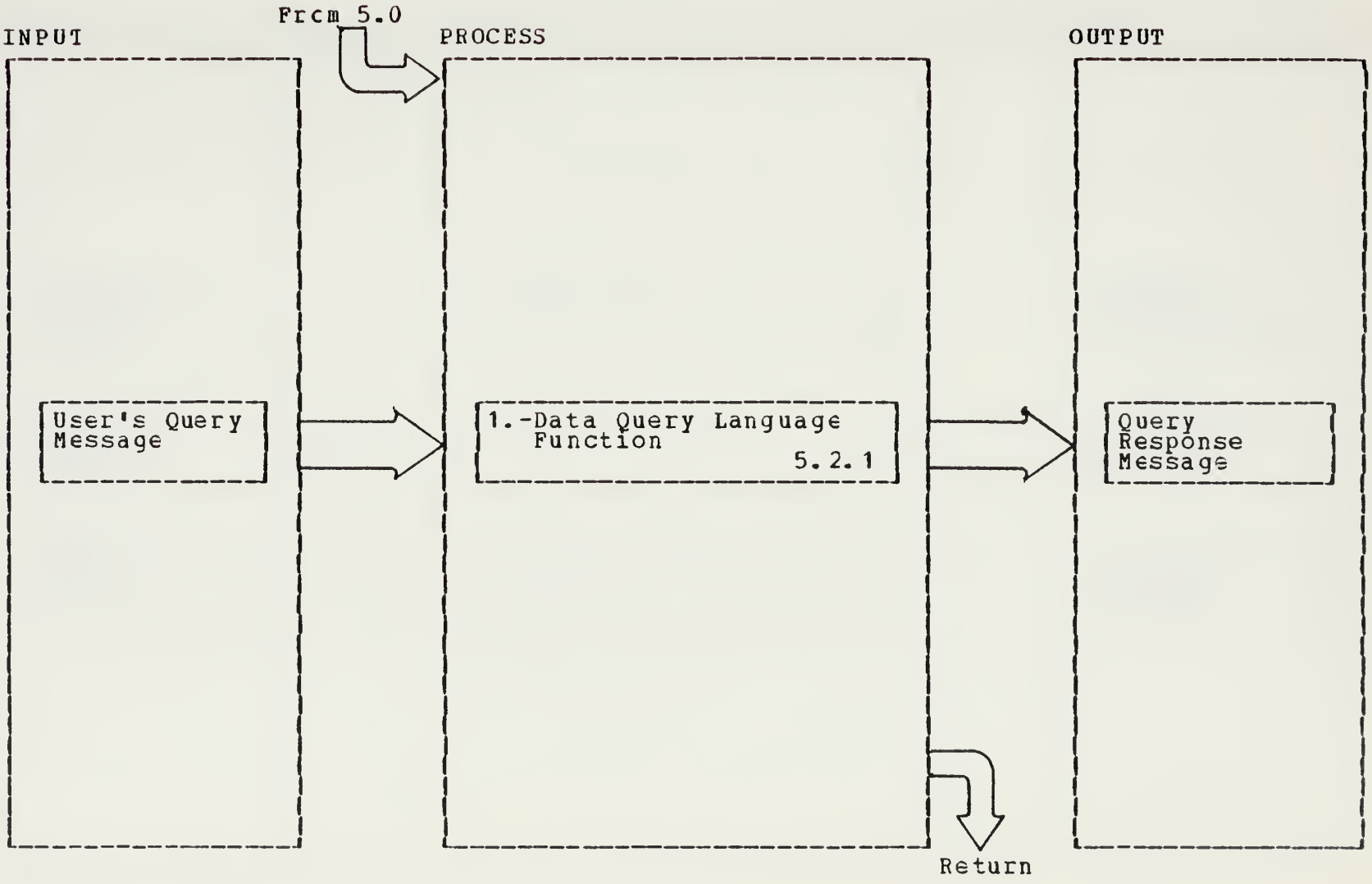
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 5.0 Name:LAN034 Description:Database Management Service



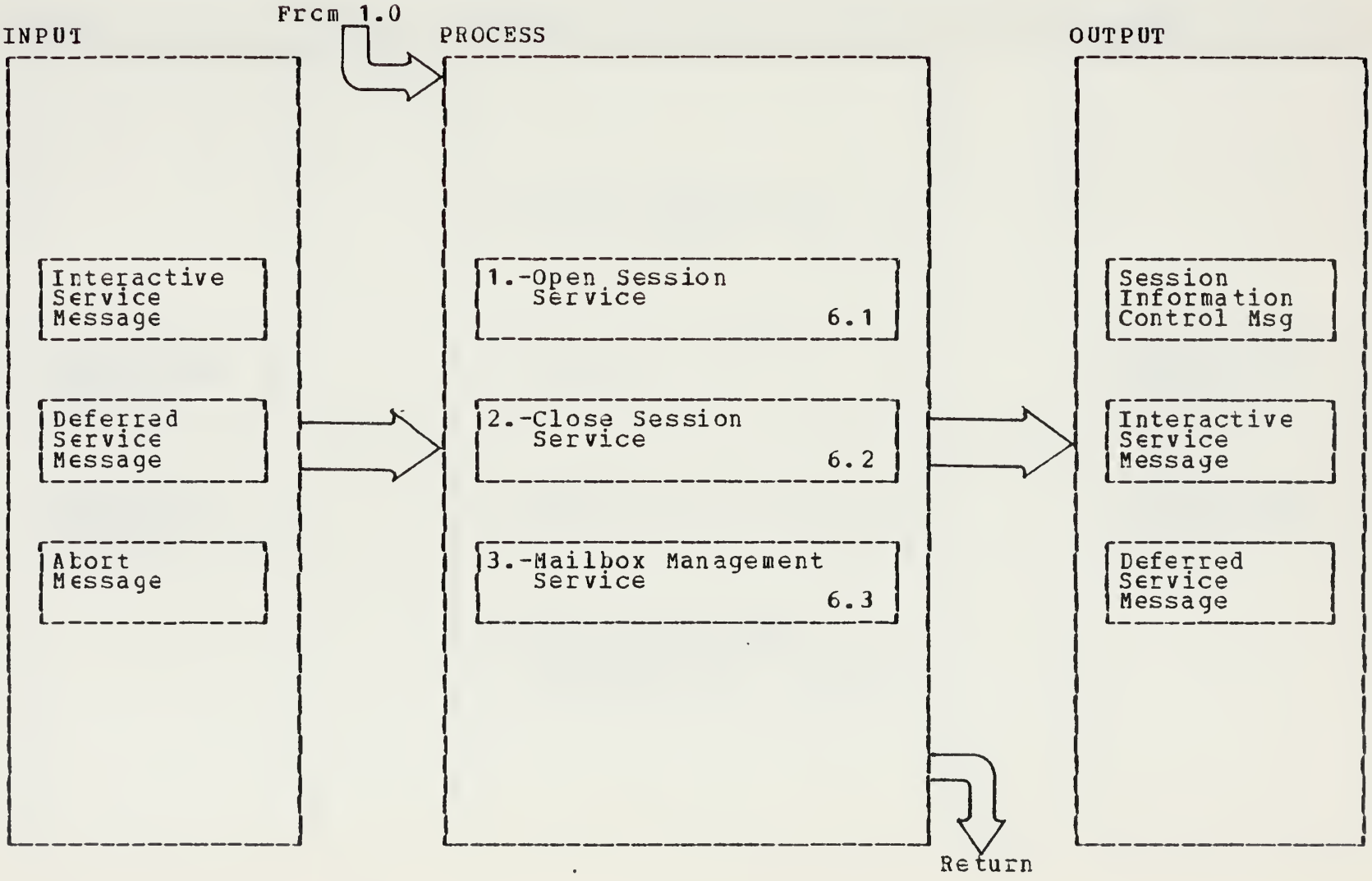
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 5.1 Name:LAN035 Description:File Management



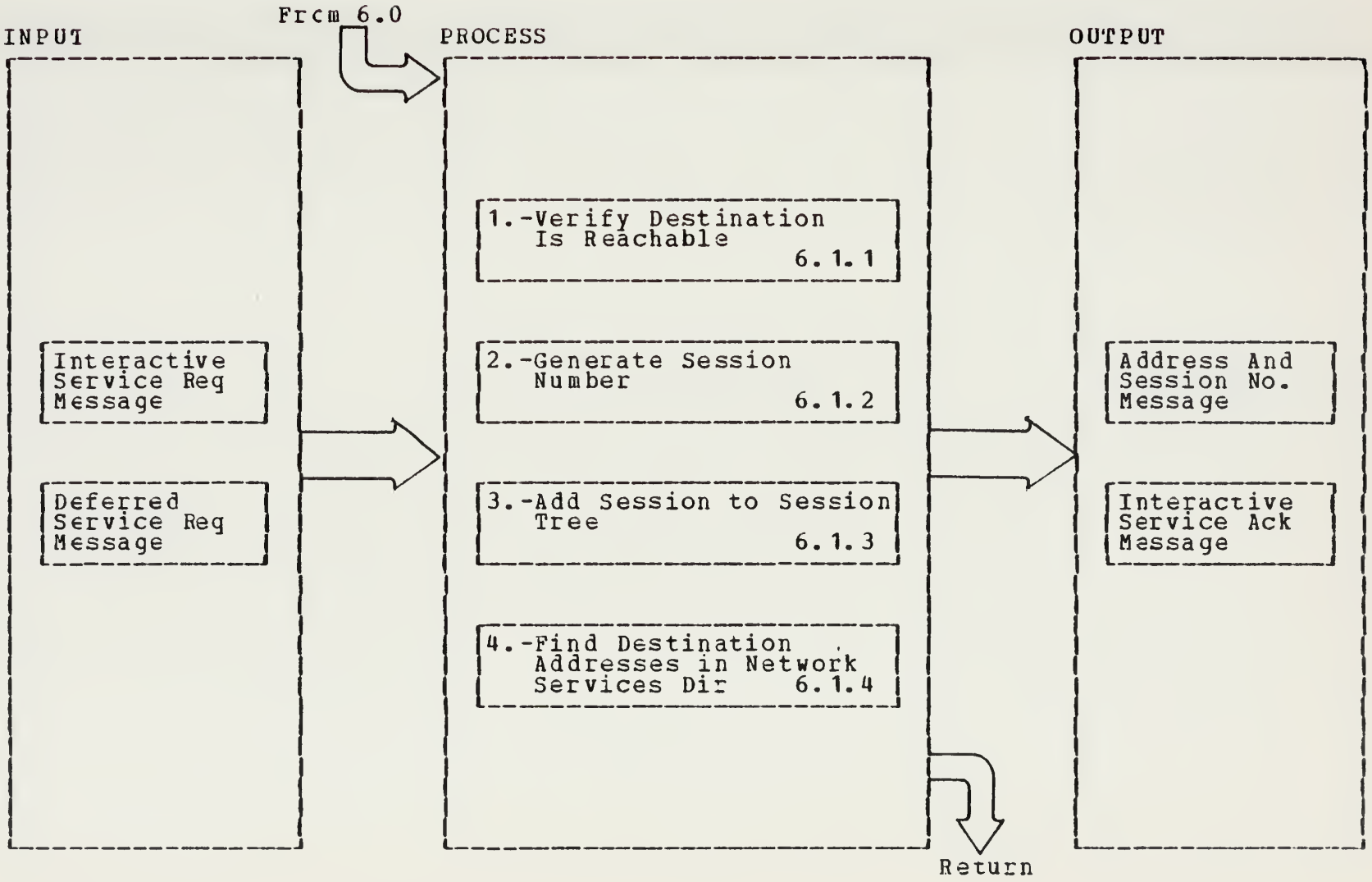
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 5.2 Name:LAN036 Description:Query Processing Service



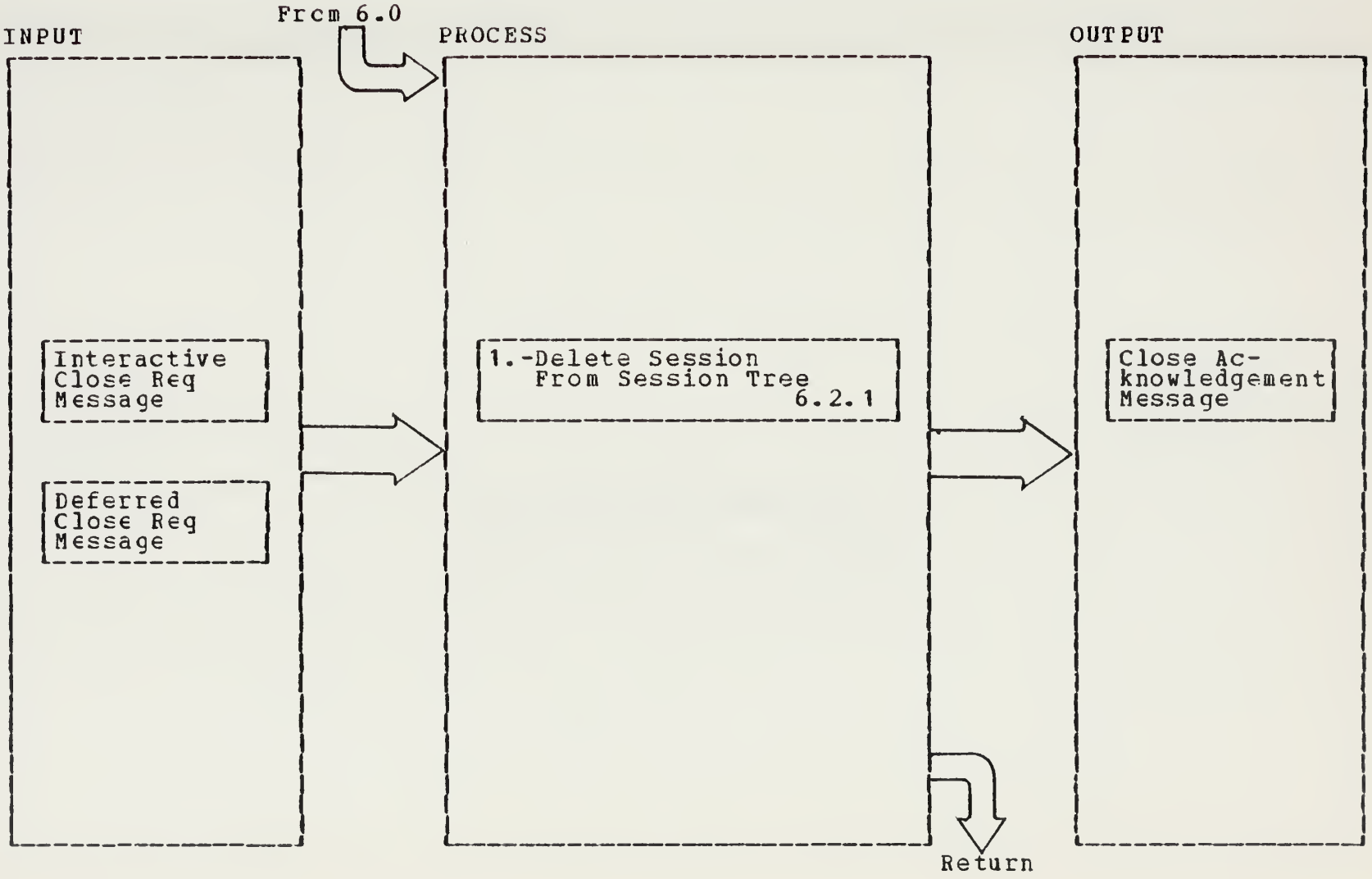
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 6.0 Name:LAN037 Description:Session Service



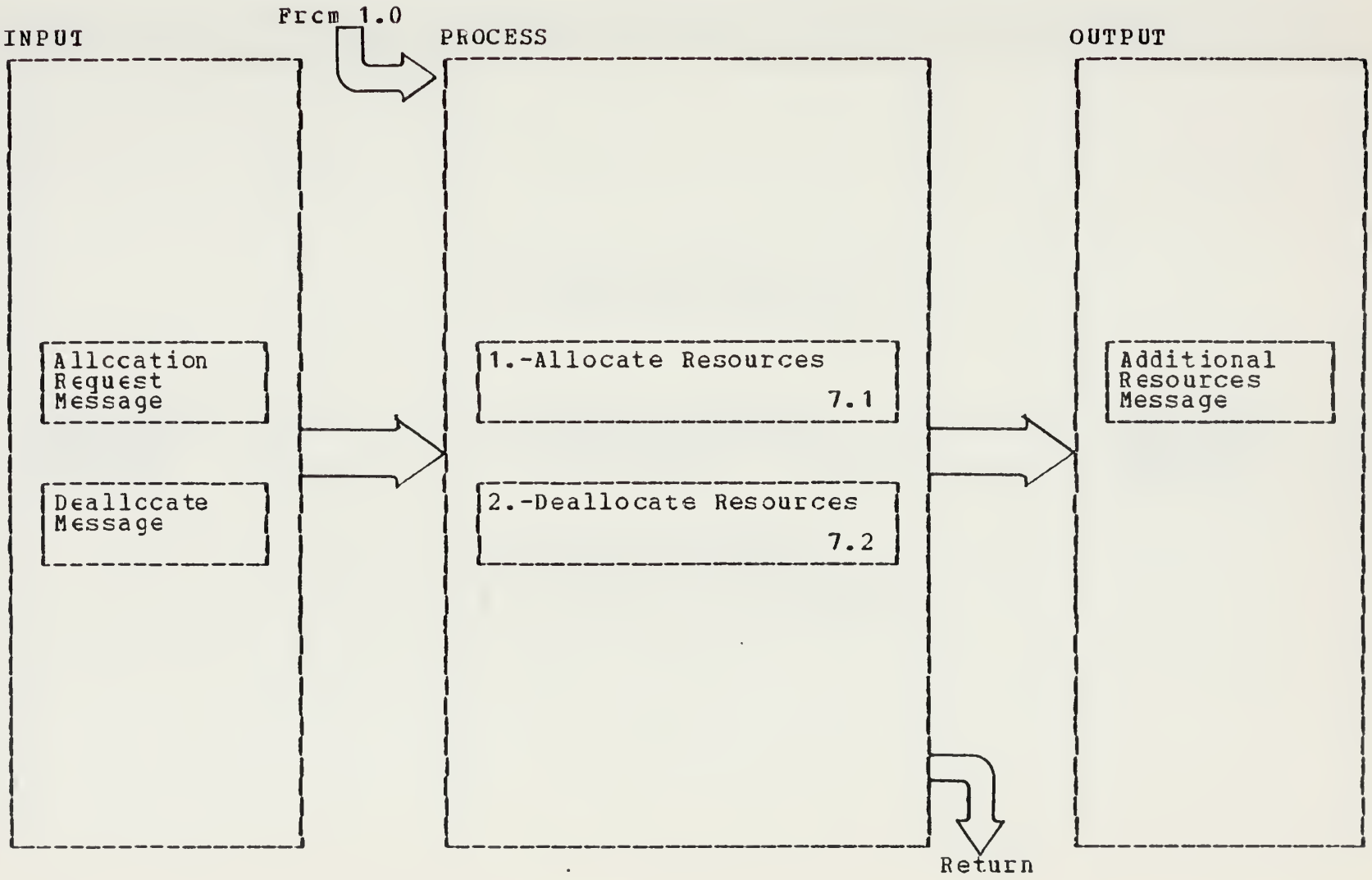
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 6.1 Name:LAN038 Description:Open Session Service



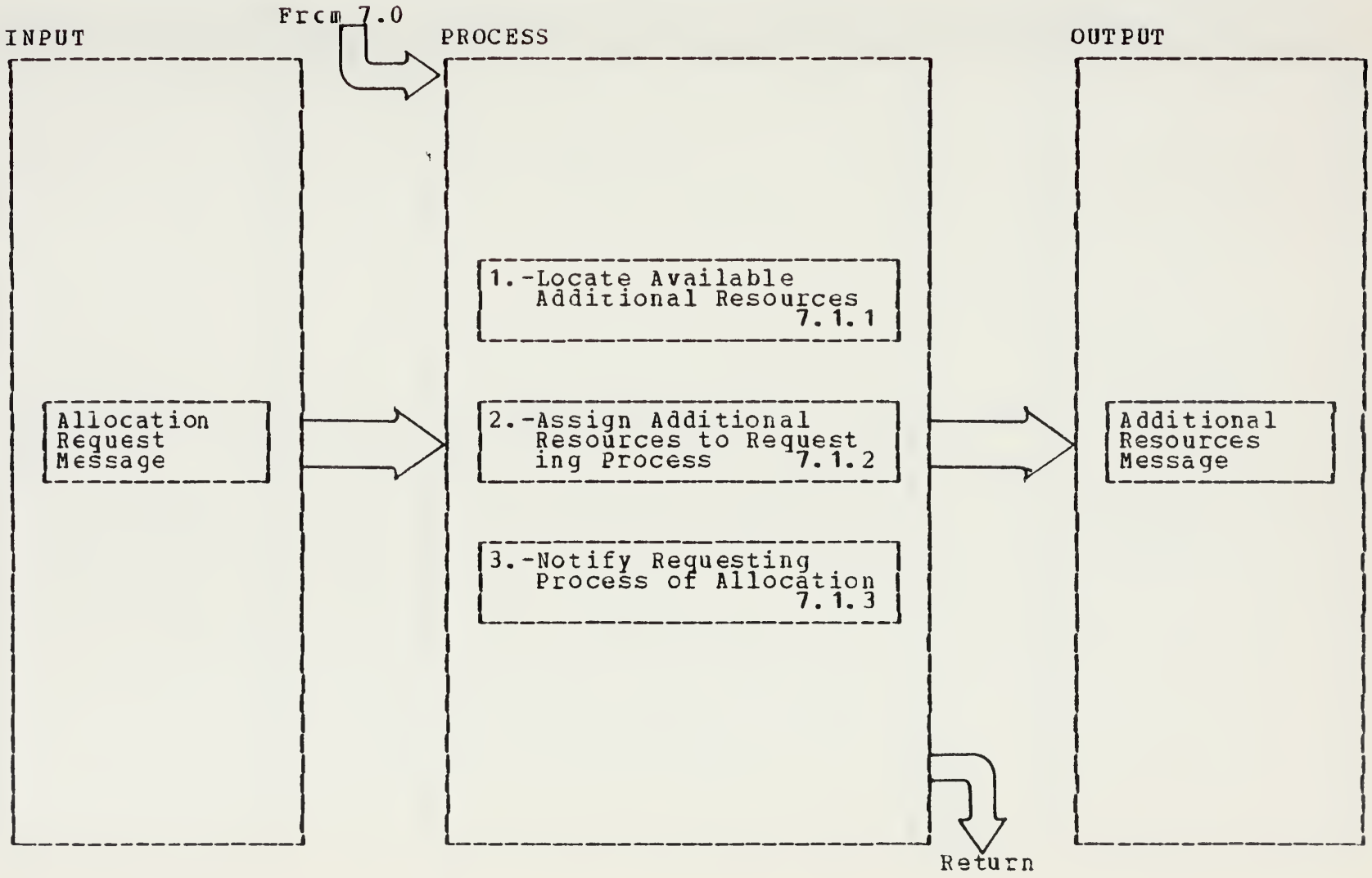
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 6.2 Name:LAN039 Description:Close Session Service



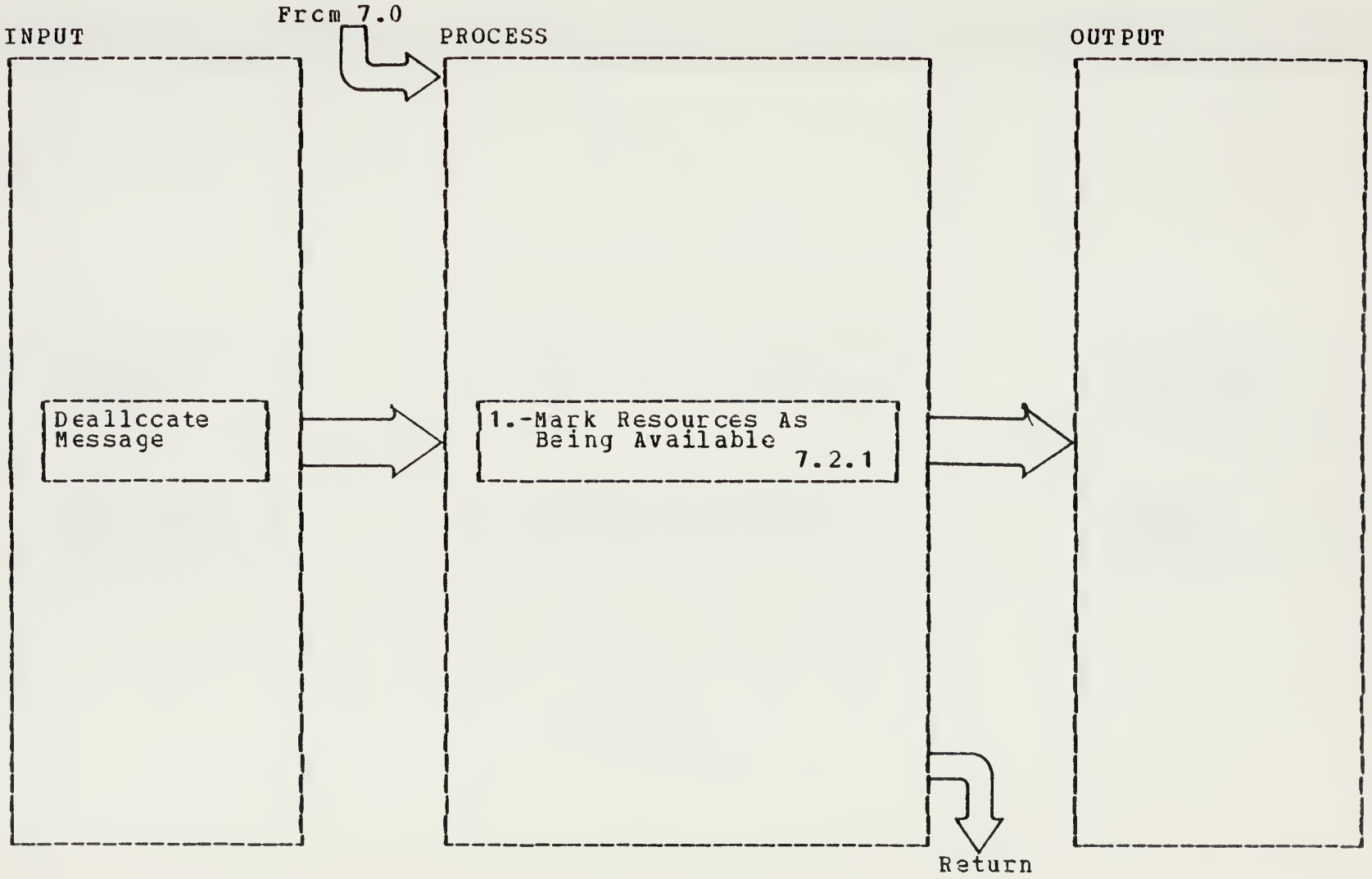
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 7.0 Name:LAN040 Description:Resource Allocation Services



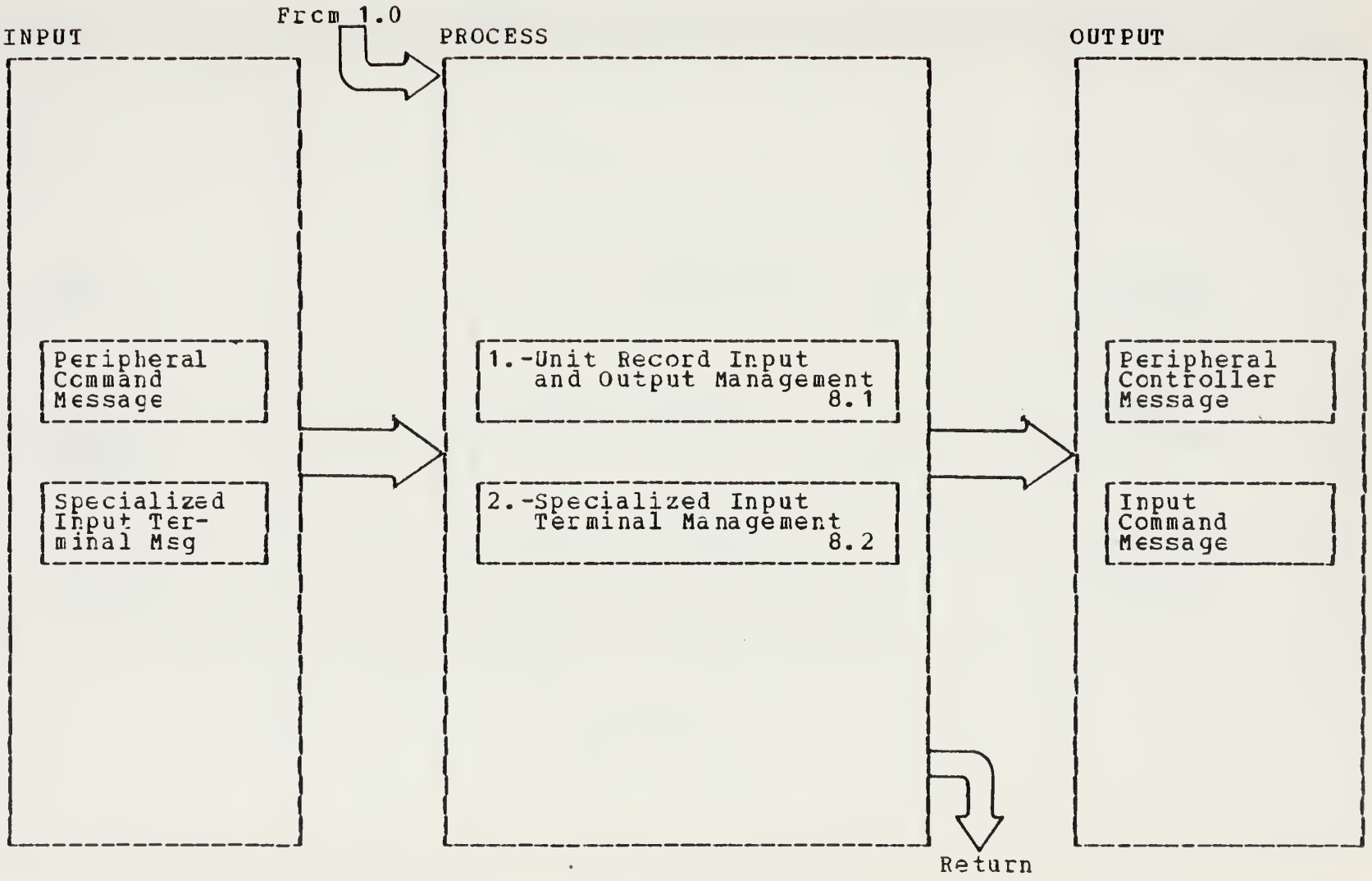
Authcr:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 7.1 Name:LAN041 Description:Allocate Resources



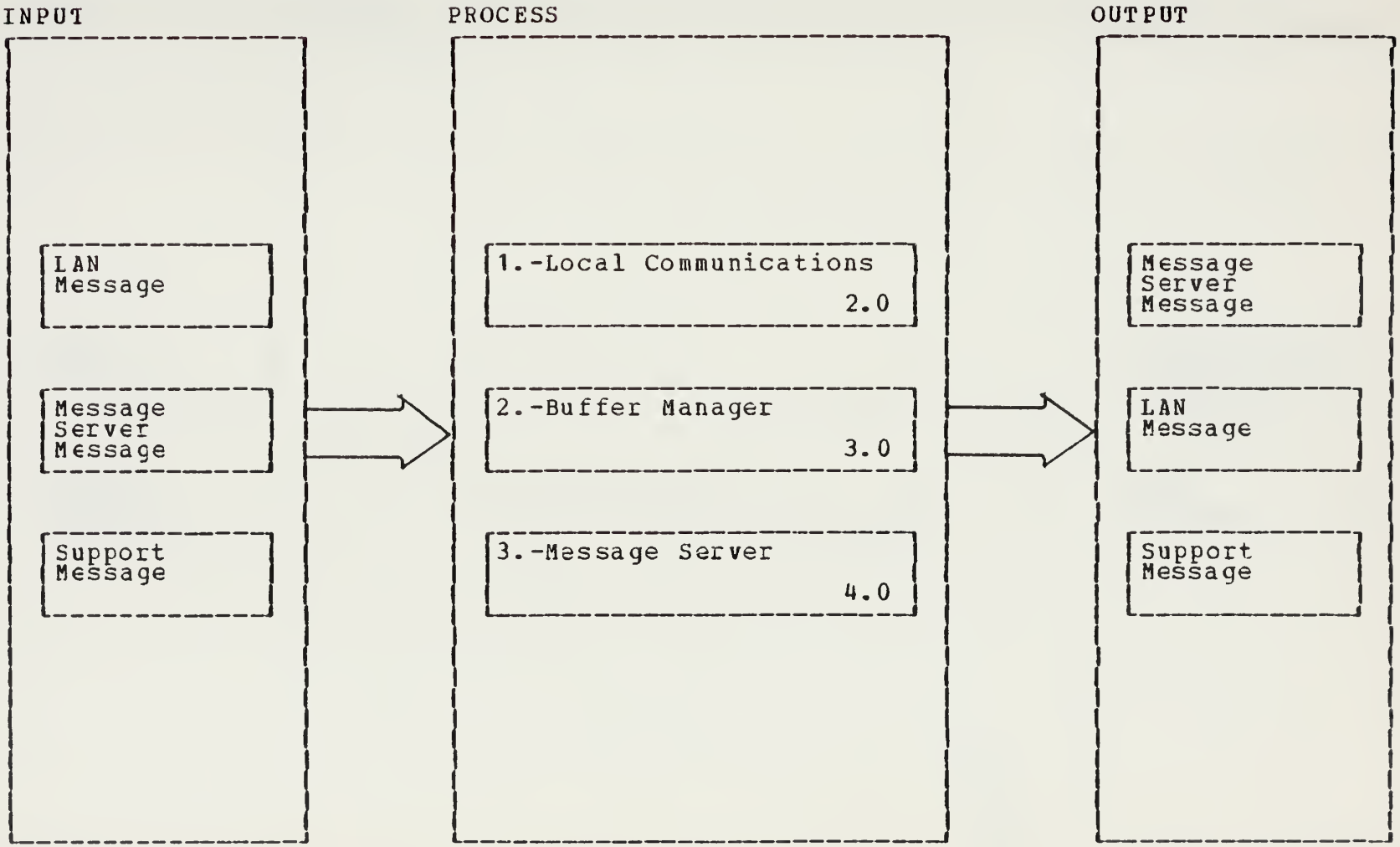
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 7.2 Name:LAN042 Description:Deallocate Resources



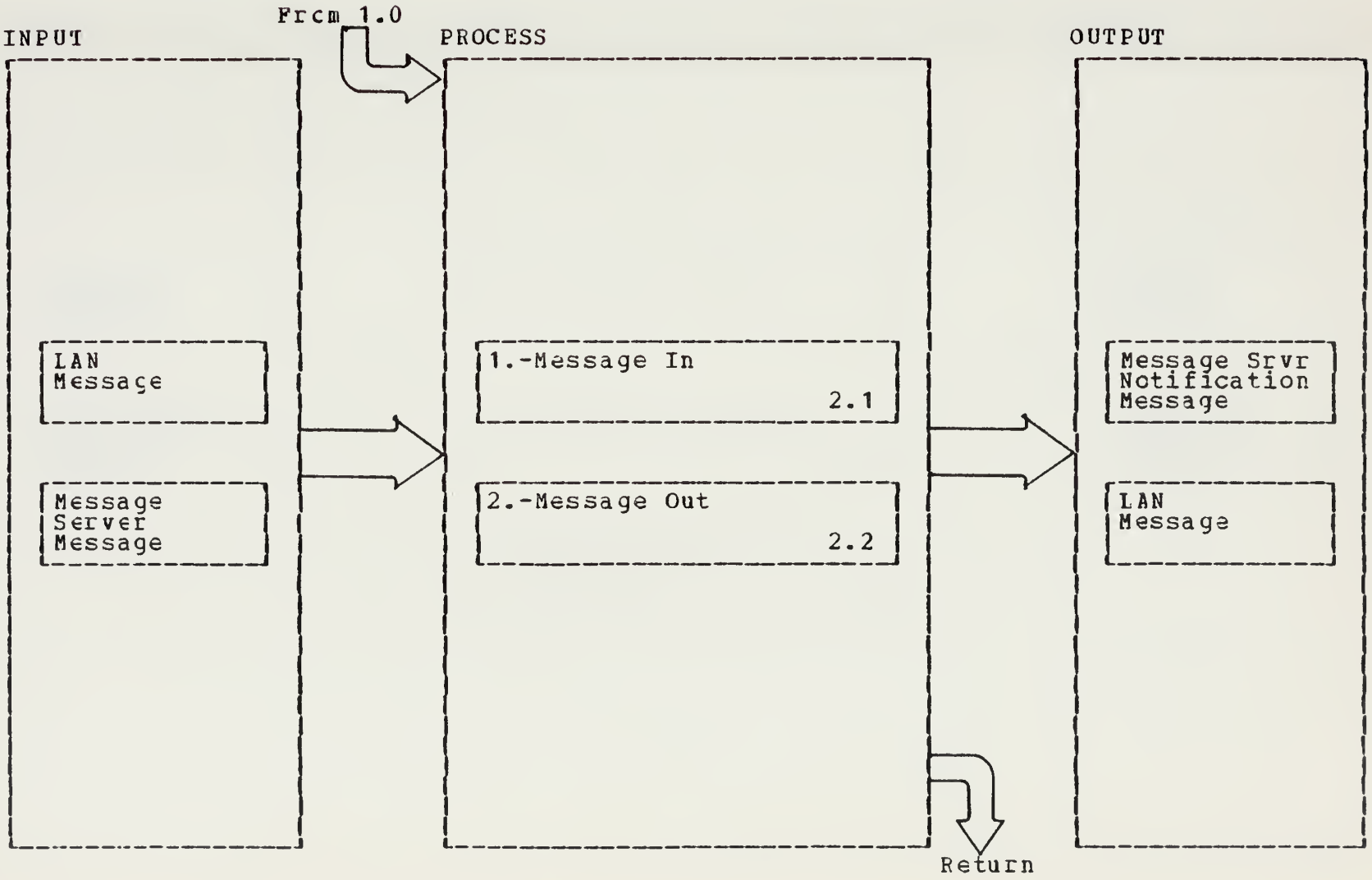
Authcr:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 8.0 Name:LAN043 Description:Peripheral Management



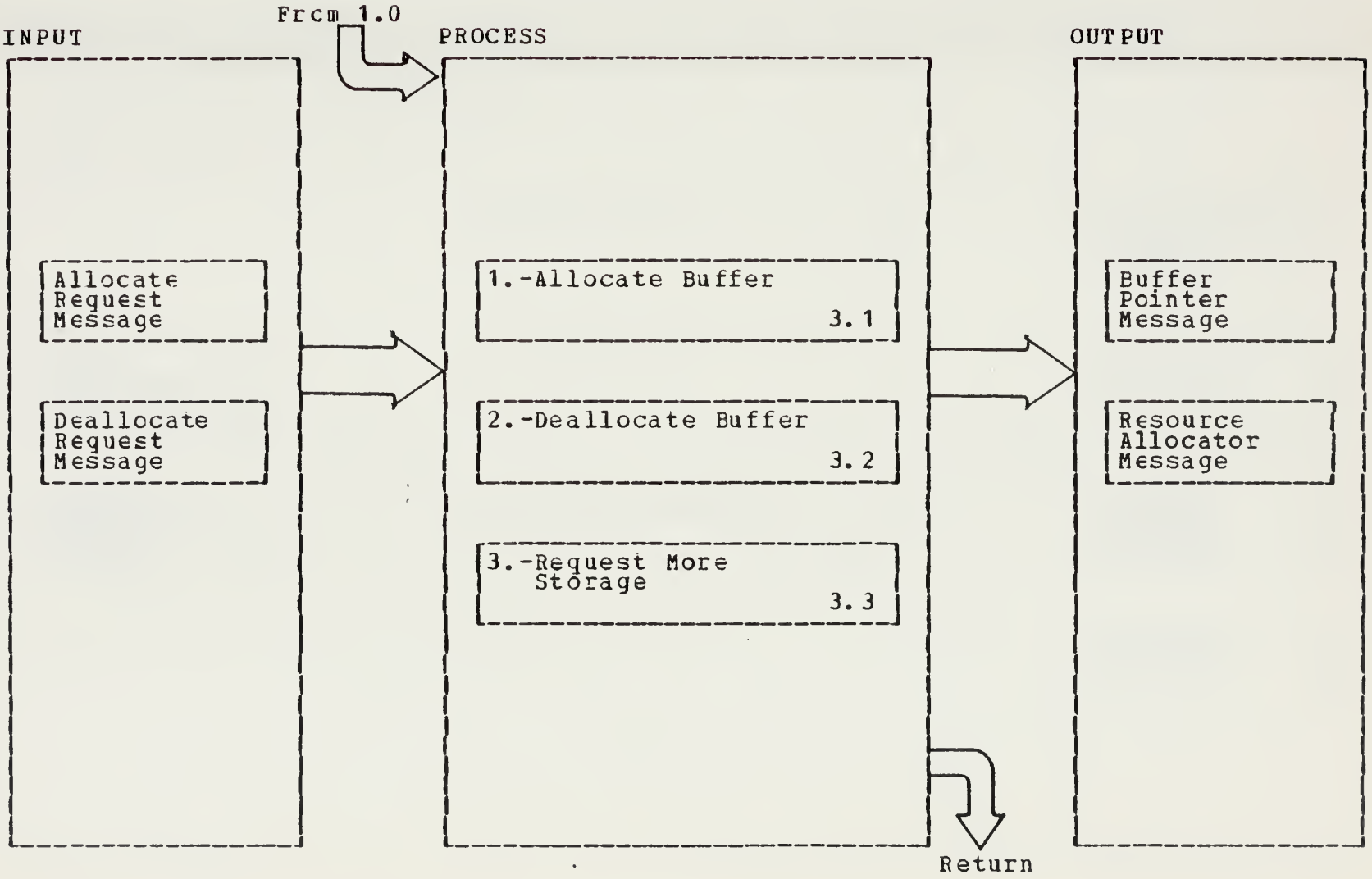
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 1.0 Name:LAN044 Description:Functional Module Support Services



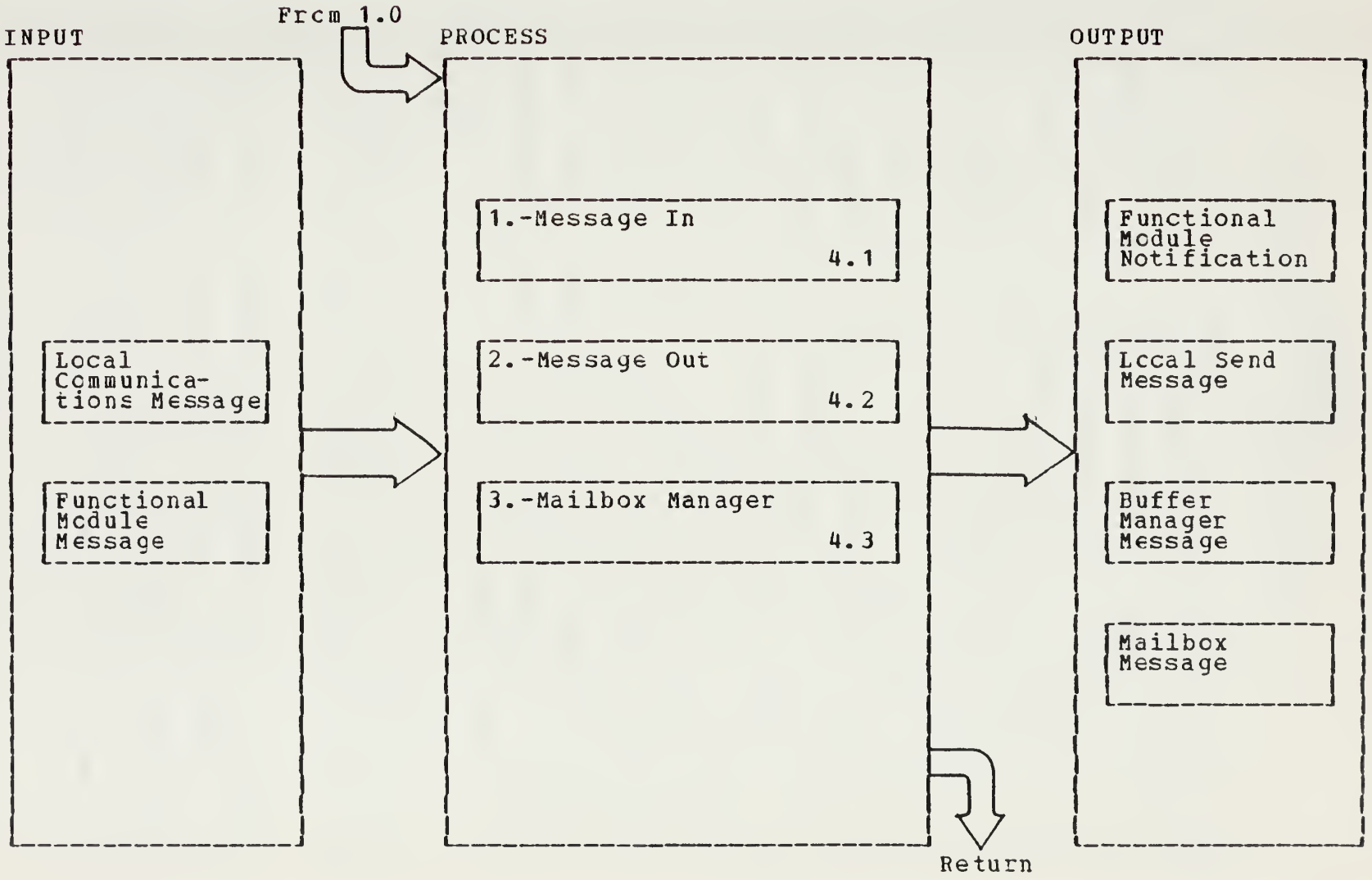
Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 2.0 Name:LAN045 Description:Local Communications



Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 3.0 Name:LAN046 Description:Buffer Manager



Author:Krebill/Carlsen System/Program:SPLICE Local Area Network Date:28 May 83
Diagram ID: 4.0 Name:LAN047 Description:Message Server



APPENDIX B
NC ALGORITHMIC REPRESENTATION

This Appendix is a pseudo code representation of the National Communications (NC) Modules. It is designed to closely conform to the HIPO diagrams in Appendix A. It provides greater detail of the sequence of actions of NC. A sequential execution is assumed throughout; however, the Message Server (MS) and TCP processes are expected to be communicating processes with NC. As such, MS or TCP can execute concurrent calls on NC, but NC is not interruptable, and will deal with each call in turn. The code does not describe any precedence handling constructs or methods due to the sequential execution assumption. To add precedence queues will require a determination of where breaks in the sequential flow should occur. At those points, the sets of precedence queues and control message queue should begin. The process servicing those queues would always choose the highest precedence message first, and would choose an interactive message before a deferred one. These queues and queue servicing mechanisms appear in the next appendix, but they are essentially provided by built-in language constructs defined for Ada.

A more detailed state diagram is shown in figure B.1 to provide an overview of the NC pseudo code. Note that NC closely follows the state changes associated with TCP connections.

This design does suggest some possible ways NC modules could be structured to obtain multitasking. For example, the submodules encapsulate states which are mutually independent of one another. A LAN session progresses through each state and can only exist in one state at a time. This implies that

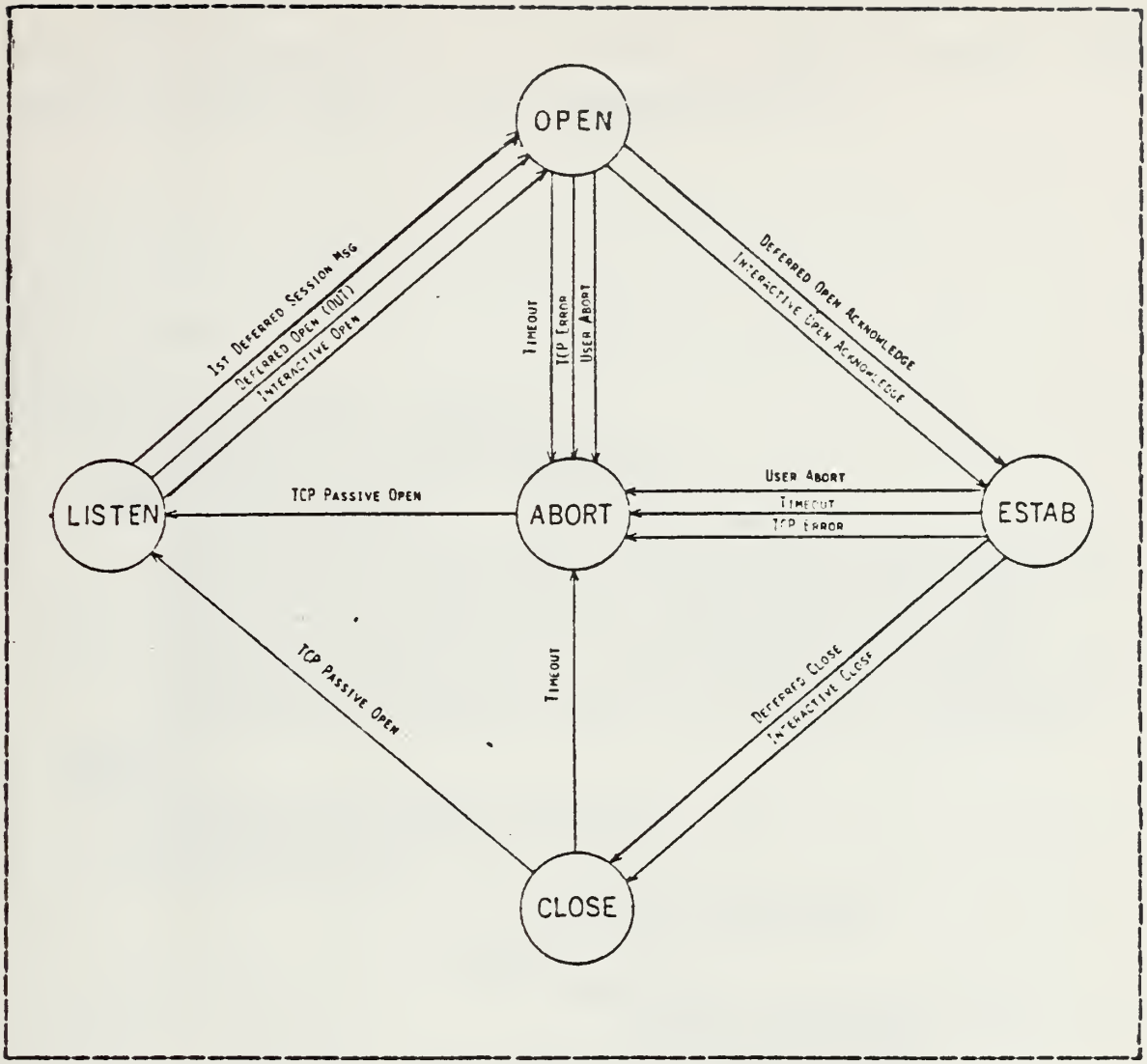


Figure B.1 NC Module Detailed State Diagram.

the lower submodules, such as OPEN.Interactive.OUT, could be made into separate processes and not merely subroutines of NC Main. In fact, multiple copies of each of the submodules--one for each session--might be implemented. Another alternative is to make all of NC reentrant. Appendix C attempts to depict NC using the Ada language as a Systems Design Language (SDL). The structure uses a great degree of concurrency based on Ada tasks.

This appendix was included so as to outline the many details associated with NC actions. It should provide a good initial design for an actual implementation.

A. NC_MAIN

```
Do While NC_Operate = True
  If Message_Server_Call then
    - Interpret LAN_MSG (MSG_TYPE)
    - Build/update Session Table Entries
    Case MSG_TYPE is
       Session_ABORT ==> Call ABORT.OUT
       Interactive_OPEN_RQST ==> Call OPEN.Interactive.OUT
       Deferred_OPEN_RQST ==> Call OPEN.Deferred.OUT
       1st_Deferred_Data_MSG ==> Call ESTAB.Deferred.OUT
       Deferred_Data_MSG ==> Call ESTAB.Deferred.OUT
       Interactive_Data_MSG ==> Call ESTAB.Interactive.OUT
       Deferred_CLOSE_MSG ==> Call CLOSE.Deferred.OUT
       Interactive_CLOSE_MSG ==> Call CLOSE.Interactive.OUT
       Otherwise ==> Report Error to Recovery Management
    End Case
  End If
  If TCP_MSG_CALL then
    - Interpret TCP_MSG (MSG_TYPE)
    - Build/update Session Table Entries
    Case MSG_TYPE is
       Error_MSG ==> If TCP Response =
                          "Connection Reset"
                      Then Call ABORT.IN
                   Else If
                      TCP Response =
                          "Connection Aborted due
                             to User Timeout"
                      Then Call Error_Handler.IN
                   Else Call Error_Handler.OUT
                   End If
       TCP_OPEN_Connection_RQST ==>  Allocate MSG_Buffer
                                     Issue TCP_Receive
       Interactive_OPEN_RQST ==> Call OPEN.Interactive.IN
       1st_Deferred_Data_MSG ==> Call OPEN.Deferred.IN
       Deferred_Data_MSG ==> Call ESTAB.Deferred.IN
       Interactive_Data_MSG ==> Call ESTAB.Interactive.IN
       Deferred_CLOSE_MSG ==> Call CLOSE.Deferred.IN
       Interactive_CLOSE_MSG ==> Call Interactive.CLOSE.IN
       Otherwise ==> Report Error to Recovery Management
    End Case
  End If
End Do
```

B. OPEN

1. Interactive

a. OPEN.Interactive.OUT

```
- Issue Active Open to TCP (Initiate Connection)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP OPEN
                   at least two more times
                   If still same error results then:
                    - Send LAN message with TCP error message
                          to user
                    - Send error report to Recovery Management
                    - Call OUT.ABORT
                    - Return
                   End If
         Otherwise ==> Continue
      End Case
- Return from TCP Active Open with Local Connection Name
- Complete Session Table Entries
- Issue Send to TCP (Interactive OPEN Request)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP SEND
                   at least two more times
                   If still same error results then:
                    - Send LAN message with TCP error message
                          to user
                    - Send error report to Recovery Management
                    - Call OUT.ABORT
                    - Return
                   End If
         Otherwise ==> Continue
      End Case
- Allocate MSG_Buffer
- Issue Receive to TCP (Prepare for Destination
  Acknowledgement)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP RECEIVE
                   at least two more times
                   If still same error results then:
                    - Deallocate Old MSG_Buffer
                    - Send LAN message with TCP error message
                          to user
                    - Send error report to Recovery Management
                    - Call OUT.ABORT
                    - Return
                   End If
         Otherwise ==> Continue
      End Case
- Await Interactive OPEN Acknowledgement (from destination
  process)
       If Not Acknowledged by Default Timeout then
          - Deallocate Old MSG_Buffer
          - Send LAN message with "NC Timeout"
                error message to user
          - Send error report to Recovery Management
          - Call OUT.ABORT
          - Return
- Return from TCP Receive (Interactive OPEN Acknowledgement)
```

```
- Issue Local Send with  Interactive OPEN Acknowledgement to
  Source Message Server
- Allocate MSG_Buffer
- Issue Receive to TCP (Prepare for Session)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP RECEIVE
                   at least two more times
                If still same error results then:
                 - Deallocate Old MSG_Buffer
                 - Send LAN message with TCP error message
                         to user
                 - Send error report to Recovery Management
                 - Call OUT.ABORT
                 - Return
                End If
         Otherwise ==> Continue
      End Case
- RETURN
```

b. OPEN.Interactive.IN

```
- Issue Local Send to Destination Message Server with
  Interactive OPEN Request (Local Connection Name)
- Receive Interactive OPEN Acknowledgement (Local Connection
  Name, Session Number, Source Addr, Destination Addr,
  Source NC Addr)
- Complete Session Table Entries
- Issue Send to TCP (Interactive OPEN Acknowledgement)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP SEND
                   at least two more times
                If still same error results then:
                 - Send LAN message with TCP error message
                         to user
                 - Send error report to Recovery Management
                 - Call OUT.ABORT
                 - Return
                End If
         Otherwise ==> Continue
      End Case
- Allocate MSG_Buffer
- Issue Receive to TCP (Prepare for next Message)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP RECEIVE
                   at least two more times
                If still same error results then:
                 - Deallocate Old MSG_Buffer
                 - Send LAN message with TCP error message
                         to user
                 - Send error report to Recovery Management
                 - Call OUT.ABORT
                 - Return
                End If
         Otherwise ==> Continue
      End Case
- RETURN
```

2. Deferred

a. OPEN.Deferred.OUT

```
- Issue Active Open to TCP (Initiate Connection)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP OPEN
                   at least two more times
                If still same error results then:
                 - Send LAN message with TCP error message
                         to user
                 - Send error report to Recovery Management
                 - Call OUT.ABORT
                 - Return
                End If
         Otherwise ==> Continue
      End Case
- Return from TCP Open (Local Connection Name)
- Complete Session Table
- Respond to Session Services' Message
- Allocate MSG_Buffer
- Issue Receive to TCP (Prepare for session message acknowl-
  edgements)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP RECEIVE
                   at least two more times
                If still same error results then:
                 - Deallocate Old MSG_Buffer
                 - Send LAN message with TCP error message
                         to user
                 - Send error report to Recovery Management
                 - Call OUT.ABORT
                 - Return
                End If
         Otherwise ==> Continue
      End Case
- RETURN
```

b. OPEN.Deferred.IN

```
- Send LAN Message to Session Services (Precedence,
  Classification, Local Connection Name, Source Addr,
  Destination Addr)
- Receive LAN Message from Session Services (Local
  Connection Name, Session Number, Source NC Addr)
- Complete entries in Session Table
- Issue send to TCP (Source NC Acknowledgement)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP SEND
                   at least two more times
                If still same error results then:
                 - Send LAN message with TCP error message
                         to user
                 - Send error report to Recovery Management
                 - Call OUT.ABORT
                 - Return
                End If
         Otherwise ==> Continue
      End Case
```

```
- Allocate MSG_Buffer
- Issue Receive to TCP (Prepare for session)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP RECEIVE
                   at least two more times
                  If still same error results then:
                   - Deallocate Old MSG_Buffer
                   - Send LAN message with TCP error message
                         to user
                   - Send error report to Recovery Management
                   - Call OUT.ABORT
                   - Return
                  End If
         Otherwise ==> Continue
      End Case
- Issue Local Send (Session Number, Destination)
- RETURN
```

C. ESTABLISHED

1. Interactive

a. ESTAB.Interactive.OUT

```
- Map Session number to Local Connection Name in Session
  Table
- Issue Send to TCP (Outgoing LAN message)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP SEND
                   at least two more times
                  If still same error results then:
                   - Deallocate Old MSG_Buffer
                   - Send LAN message with TCP error message
                         to user
                   - Send error report to Recovery Management
                   - Call OUT.ABORT
                   - Return
                  End If
         Otherwise ==> Continue
      End Case
- Await Acknowledgement from Remote NC
        If Not Acknowledged by Default Timeout then
           - Deallocate Old MSG_Buffer
           - Send LAN message with "NC Timeout"
             error message to user
           - Send error report to Recovery Management
           - Call OUT.ABORT
           - Return
        End If
- Update Session Table
- Obtain MSG_Buffer
- Issue Receive to TCP (prepare for next message)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP RECEIVE
                   at least two more times
                  If still same error results then:
                   - Deallocate Old MSG_Buffer
                   - Send LAN message with TCP error message
```

```
                        to user
                    - Send error report to Recovery Management
                    - Call OUT.ABORT
                    - Return
                   End If
          Otherwise ==> Continue
      End Case
- RETURN
```

b. ESTAB.Interactive.IN

```
- Map Local Connection Name to Session Number
- Verify sequence number of fragment/message
- Issue Send to TCP (Source NC Acknowledgement)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP SEND
                   at least two more times
                   If still same error results then:
                    - Send LAN message with TCP error message
                        to user
                    - Send error report to Recovery Management
                    - Call OUT.ABORT
                    - Return
                   End If
          Otherwise ==> Continue
      End Case
- Obtain MSG_Buffer
- Issue Receive to TCP (prepare for next message)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP RECEIVE
                   at least two more times
                   If still same error results then:
                    - Deallocate Old MSG_Buffer
                    - Send LAN message with TCP error message
                        to user
                    - Send error report to Recovery Management
                    - Call OUT.ABORT
                    - Return
                   End If
          Otherwise ==> Continue
      End Case
- Update Session Table
- Issue Local Send (new message number assigned)  to Message
  Server
- RETURN
```

2. Deferred

a. ESTAB.Deferred.OUT

```
- Map Session number to Local Connection Name in Session
  Table
- Issue Send to TCP (Outgoing LAN message)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP SEND
                   at least two more times
                   If still same error results then:
                    - Deallocate Old MSG_Buffer
```

```
                - Send LAN message with TCP error message
                    to user
                - Send error report to Recovery Management
                - Call OUT.ABORT
                - Return
              End If
        Otherwise ==> Continue
    End Case
- Await Acknowledgement from Remote NC
      If Not Acknowledged by Default Timeout then
         - Deallocate Old MSG_Buffer
         - Send LAN message with "NC Timeout"
           error message to user
         - Send error report to Recovery Management
         - Call OUT.ABORT
         - Return
      End If
- Obtain MSG_Buffer
- Issue Receive to TCP (Prepare for next message)
- Await TCP Signal (Response)
    Case Response is
       "Insufficient Resources"
            ==> Attempt TCP RECEIVE
                at least two more times
             If still same error results then:
              - Deallocate Old MSG_Buffer
              - Send LAN message with TCP error message
                    to user
              - Send error report to Recovery Management
              - Call OUT.ABORT
              - Return
             End If
       Otherwise ==> Continue
    End Case
- RETURN
```

b. ESTAB.Deferred.IN

```
- Map Local Connection Name to Session Number
- Verify sequence number of fragment/message
- Issue Send to TCP (Source NC Acknowledgement)
- Await TCP Signal (Response)
    Case Response is
       "Insufficient Resources"
            ==> Attempt TCP SEND
                at least two more times
             If still same error results then:
              - Send LAN message with TCP error message
                    to user
              - Send error report to Recovery Management
              - Call OUT.ABORT
              - Return
             End If
       Otherwise ==> Continue
    End Case
- Obtain MSG_Buffer
- Issue Receive to TCP (prepare for next message)
- Await TCP Signal (Response)
    Case Response is
       "Insufficient Resources"
            ==> Attempt TCP RECEIVE
                at least two more times
             If still same error results then:
              - Deallocate Old MSG_Buffer
              - Send LAN message with TCP error message
                    to user
              - Send error report to Recovery Management
```

```
                - Call OUT.ABORT
                - Return
              End If
        Otherwise ==> Continue
    End Case
- Update Session Table
- Issue Local Send (new message number assigned)  to Message
  Server
- RETURN
```

D. CLOSE

1. Interactive

a. CLOSE.Interactive.OUT

```
- Issue Send to TCP (Interactive Close Request)
- Await Acknowledgement from Remote NC
      If Not Acknowledged by Default Timeout then
         - Send LAN message with "NC Timeout"
           error message to user
         - Send error report to Recovery Management
         - Call OUT.Abort
         - Return
      End If
- Return    from    TCP    Receive    (Interactive    Close
  Acknowledgement)
- Issue Local Send with Interactive Close Acknowledgement to
  Session Services or Source Message Server
- Issue Close to TCP (Terminate Local Connection)
- Delete Entries in Session Table
- Issue Passive Open to TCP (Prepare for new Connections)
- Await TCP Signal (Response)
    Case Response is
      "Insufficient Resources"
            ==> Attempt TCP Passive OPEN
                at least two more times
             If still same error results then:
              - Send LAN message with TCP error message
                     to user
              - Send error report to Recovery Management
              - Return
             End If
      Otherwise ==> Continue
    End Case
- RETURN
```

b. CLOSE.Interactive.IN

```
- Issue Send to TCP (Source NC Acknowledgement)
- Await TCP Signal (Response)
    Case Response is
      "Insufficient Resources"
            ==> Attempt TCP SEND
                at least two more times
             If still same error results then:
              - Send LAN message with TCP error message
                     to user
              - Call OUT.Abort
              - Send error report to Recovery Management
              - Return
             End If
```

```
         Otherwise ==> Continue
      End Case
- Issue Close to TCP (Terminate Local connection)
- Send LAN Message to Session Services to delete Session
- Delete Entries from Session Table
- Issue Passive Open to TCP (Prepare for new Connections)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP Passive OPEN
                   at least two more times
                If still same error results then:
                 - Send LAN message with TCP error message
                       to user
                 - Send error report to Recovery Management
                 - Return
                End If
         Otherwise ==> Continue
      End Case
- RETURN
```

2. Deferred

a. CLOSE.Deferred.OUT

```
- Issue TCP Send (Deferred Session Close Message)
- Return from TCP Receive (NC Close Acknowledgment)
- Issue Close to TCP (Terminate Local Connection)
- Delete entries on Session Table
- Issue Passive Open to TCP (Prepare for new Connections)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP Passive OPEN
                   at least two more times
                If still same error results then:
                 - Send LAN message with TCP error message
                       to user
                 - Send error report to Recovery Management
                 - Return
                End If
         Otherwise ==> Continue
      End Case
- RETURN
```

b. CLOSE.Deferred.IN

```
- Issue send to TCP (Source NC Acknowledgement)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP SEND
                   at least two more times
                If still same error results then:
                 - Send LAN message with TCP error message
                       to user
                 - Send error report to Recovery Management
                 - Return
                End If
         Otherwise ==> Continue
      End Case
- Issue Close to TCP (Terminate Local Connnection)
- Send LAN Message to Session Services to delete Session
- Delete entries in Session Table
```

```
- Issue Passive Open to TCP (Prepare for new Connections)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP Passive OPEN
                   at least two more times
                If still same error results then:
                 - Send LAN message with TCP error message
                         to user
                 - Send error report to Recovery Management
                 - Return
                End If
         Otherwise ==> Continue
      End Case
- RETURN
```

E. ABORT

1. ABORT.OUT

```
- Purge Session Message Fragment Queue
- Issue ABORT to TCP
- Delete entries on Session Table
- Issue Passive Open to TCP (Prepare for new Connections)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP Passive OPEN
                   at least two more times
                If still same error results then:
                 - Send LAN message with TCP error message
                         to user
                 - Send error report to Recovery Management
                 - Return
                End If
         Otherwise ==> Continue
      End Case
- RETURN
```

2. ABORT.IN

```
- Purge Session Message Fragment Queue
- Send Control Message to Session Services to ABORT Session
- Delete entries in Session Table
- Issue Passive Open to TCP (Prepare for new Connections)
- Await TCP Signal (Response)
      Case Response is
         "Insufficient Resources"
               ==> Attempt TCP Passive OPEN
                   at least two more times
                If still same error results then:
                 - Send LAN message with TCP error message
                         to user
                 - Send error report to Recovery Management
                 - Return
                End If
         Otherwise ==> Continue
      End Case
- RETURN
```

F. ERROR_HANDLER

1. Error_Handler.OUT

```
Case TCP Error Response is:
          "Precedence not allowed" or
          "Security/compartment not allowed" or
          "Connection illegal for this process"
               ==> Send LAN message with TCP error message
                        to user
                 - Initiate NC OUT Abort
          "Destination Unreachable"
                ==> Send LAN message with TCP error message
                          to user
                  - Initiate NC OUT Abort
          "Insufficient Resources"
                ==> Repeat TCP invocation (OPEN,RECEIVE or
                       SEND)  by  Calling appropriate  Routine
End Case
- Return
```

2. Error_Handler.IN

```
- Send LAN Message with TCP error message to local user
- Initiate NC IN Abort
- RETURN
```

APPENDIX C
NC DESIGN IN AN ADA SDL

This Appendix utilizes the Ada programming language as a Systems Design Language (SDL) to express a possible structuring of the National Communications (NC) and other process modules. One of the key reasons for this appendix is to suggest ways of building a multitasking model of NC and other communicating processes. As such, the Ada SDL model is not meant to precisely define all of the details, such as specific parameters of the interfaces.

Using Ada as a SDL allows one to build a multitasking design that is not dependent upon the use of specific operating systems' synchronization primitives. The design is intended to avoid using any system dependent facilities, and for that reason, the pragma priority was not used in conjunction with precedence handling. The tasking constructs of Ada readily allow one to build precedence handling queues which are needed in several SPLICE modules. Server tasks were used to make, for example, the task NATIONAL_COMMUNICATIONS asynchronous from other tasks, such as an instance of a task type OPEN_SERVER_QUEUE.

Provided in Figure C.1 is a graphic representation of the key tasks in the design. Figure C.1 is patterned after the style espoused in [Ref. 17]. It is visualized that tasks concerned with enforcing precedence will utilize select with guard constructs, and extensive use of the count attribute to check through various entry queues. Also, one might expect that most of the tasks, especially those instantiated from the task types, would have timeout with terminate alternatives. With the right utilization, only the minimum number of tasks would be active at any time.

One must note, however, that there are a number of unanswered questions concerning the efficiency of Ada tasks. It was not the purpose of this appendix to force the actual implementation of NC to be done in Ada. Rather, Ada was used as a SDL to provide a very flexible, system independent means for addressing a multitasking scheme for the NC module and its submodules. Ada SDL can provide a consistent and well defined tool for expressing designs that may ultimately be coded in a considerably different target language.

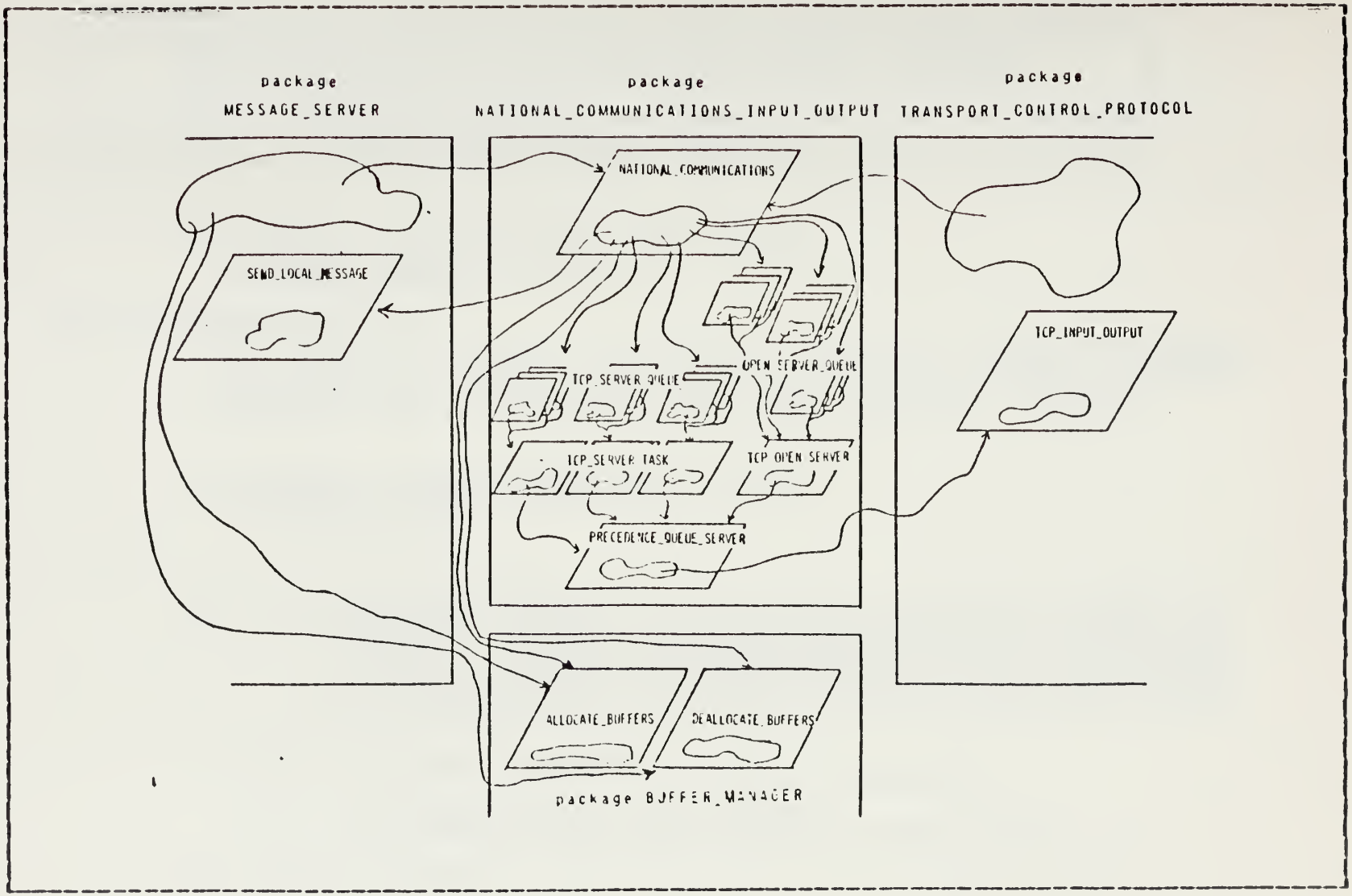


Figure C.1 NC Task Graphic Representation.

```
-- ***************************************************************
-- *   SPIICE Module Specification using ADA PACKAGE Specification   *
-- *   Contructs. Done by Dan P. Krebill and David D. Carlsen        *
-- *   for the design of the NATIONAL COMMUNICATIONS and other       *
-- *   module interfacing modules.                                   *
-- ***************************************************************
-- *                Last Update: 2 June 1983 DPK                   *
-- ***************************************************************

-------------------------------------------------------------------------------

        with NATIONAL_COMMUNICATIONS_INPUT_OUTPUT,
             TRANSPORT_CONTROL_PROTOCOL,
             MESSAGE_SERVER;

package BUFFER_MANAGER is
        -- BUFFER_MANAGER module provides the underlying storage management
        -- for message buffering. This is done explicitly through rendezvous
        -- by requesting modules to Allocate and Deallocate buffers. BUFFER_
        -- MANAGER should be capable of obtaining additional temporary
        -- resources from Resource Allocator as required.


        use NATIONAL_COMMUNICATIONS_INPUT_OUTPUT,
            TRANSPORT_CONTROL_PROTOCOL,
            MESSAGE_SERVER;


        task ALLOCATE_BUFFERS is
                -- This task manages the allocation of message buffers and
                -- provides a buffer pointer to processes (such as MESSAGE_
                -- SERVER or NATIONAL_COMMUNICATIONS). Any task performing
                -- a rendezvous with this task must provide his identity to
                -- ALLOCATE_BUFFERS so that this task can return the buffer
                -- pcinter to the proper task.

            entry ALLOCATE_TCP_BUFFER
                    (RE_ENTRY_NAME:in TCP_SERVER_TASK_NAMES);

            entry ALLOCATE_MESSAGE_SERVER_BUFFER
                    (RE_ENTRY_NAME:in MESSAGE_SERVER_TASK_NAMES);

        end ALLOCATE_BUFFERS;

        task body ALLOCATE_BUFFERS is separate;

--
```

```
        task DEALLOCATE_BUFFERS is
                -- This task allows for explicit deallocation of message
                -- buffers after a particular message/message fragment is
                -- no longer needed. This will in almost all cases be done
                -- by either MESSAGE_SERVER or NATIONAL_COMMUNICATIONS tasks.

             entry DEALLOCATE_MESSAGE_BUFFER
                     (BUFFER_POINTER:in BUFFER_ADDRESS);

        end DEALLOCATE_BUFFERS;

        task body DEALLOCATE_BUFFERS is separate;

end BUFFER_MANAGER;

package body BUFFER_MANAGER is separate;

-------------------------------------------------------------------------
```



```
        with NATIONAL_COMMUNICATIONS_INPUT_OUTPUT,
             MESSAGE_FORMATS;

package TRANSPORT_CONTROL_PROTOCOL is
        -- This package represents the specification of the Transport Control
        -- Protocol as defined in RFC 792, with the underlying assumption
        -- that TCP has some sort of tasking capability. The National
        -- Communications rendezvous/interface is designed to serialize the
        -- flow of entries/calls upon TCP, and rendezvous from TCP with NC
        -- are assumed to be serialized as well. This design somewhat
        -- supports a configuration where TCP is implemented on an outboard
        -- processor, and is interfaced via a bus or communications port.

        use NATIONAL_COMMUNICATIONS_INPUT_OUTPUT,
            MESSAGE_FORMATS;

        task TCP_INPUT_OUTPUT is
                -- This task encapsulates the basic TCP commands/calls.
                -- It is visualized that these entries will in turn
                -- generate requirements for TCP to rendezvous with NC
                -- and pass the appropriate parameters in a fashion
                 -- as described in the RFC documentation.

                  entry TCP_SEND
                          (LOCAL_PORT:in TRANSMISSION_CONTROL_BLOCK.
```

```
                                                        LOCAL_SOCKET;
                        BYTE_COUNT:in LAN_DATA_MESSAGE.LENGTH;
                        PUSH_FLAG:in TCP_HEADER_MESSAGE.PSH := NOT_PUSHED;
                        URGENT_FLAG:in TCP_HEADER_MESSAGE.URG := NOT_URGENT;
                        TIME_OUT:in TIME_TO_LIVE := DEFAULT_LIFE);

                entry TCP_OPEN
                        (LOCAL_PORT:in TRANSMISSION_CONTROL_BLOCK.
                                                        LOCAL_SOCKET;
                         FOREIGN_SOCKET:in TRANSMISSION_CONTROL_BLOCK.
                                                      FOREIGN_SOCKET;
                             TYPE_OPEN:in ACTIVE_PASSIVE_FLAG := ACTIVE;
                         TIME_OUT:in TIME_TO_LIVE := DEFAULT_LIFE;
                         CLASSIFICATION:in TRANSMISSION_CONTROL_BLOCK.
                                                  SECURITY := UNCLASSIFIED;
                             PRECEDENCE:in TRANSMISSION_CONTROL_BLOCK.
                                                    PRECEDENCE := ROUTINE;
                             OPTIONS:in TCP_HEADER_MESSAGE.OPTIONS := NO_OPTIONS);

                  entry TCP_RECEIVE
                          (LOCAL_CONNECTION:in LOCAL_CONNECTION_NAMES;
                           BUFFER_POINTER:in BUFFER_ADDRESS);

                entry TCP_CLOSE
                          (LOCAL_CONNECTION:in LOCAL_CONNECTION_NAMES);

                entry TCP_ABORT
                        (LOCAL_CONNECTION:in LOCAL_CONNECTION_NAMES);

        end TCP_INPUT_OUTPUT;

        task body TCP_INPUT_OUTPUT is separate;

end TRANSPORT_CONTROL_PROTOCOL;

package body TRANSPORT_CONTROL_PROTOCOL is separate;
-------------------------------------------------------------------------

      with  NATIONAL_COMMUNICATIONS_INPUT_OUTPUT,
            LOCAL_COMMUNICATIONS,
            BUFFER_MANAGER,
            MESSAGE_FORMATS;

package MESSAGE_SERVER is
        -- This package represents an abstraction of a common requirement
        -- for all SPLICE PMs who have need to send or receive messages.
```

```
        -- MESSAGE_SERVER should provide a FM with the capabilities to
        -- interface with LOCAL_COMMUNICATIONS and provide for error-free,
        -- sequenced delivery between any two FMs within a LAN. This also
        -- requires MESSAGE_SERVER to handle local acknowledgements for the
        -- FM. The MESSAGE_SERVER (M_S) will maintain precedence queues
        -- and will be structured to provide preference towards Interactive
        -- sessions over Deferred sessions. NATIONAL_COMMUNICATIONS tasks
        -- will interface with M_S in roughly the same way as with TCP tasks.
        -- As is it visualized that M_S will be implemented in the same
        -- physical processor as NC, no acknowledgement between them is
        -- deemed necessary.

        use NATIONAL_COMMUNICATIONS_INPUT_OUTPUT,
            LOCAL_COMMUNICATIONS,
            BUFFER_MANAGER,
            MESSAGE_FORMATS;


        task SEND_LOCAL_MESSAGE is
                -- This task is used by NC to forward a message to a local
                -- addressee. Guaranteed delivery to either the addressee
                -- or his mailbox is assumed. Control messages should be
                -- handled prior to Data messages.

                  entry SEND_LOCAL_DATA_MESSAGE
                              (BUFFER_POINTER:in BUFFER_ADDRESS;
                                PRECEDENCE:in LAN_DATA_MESSAGE.PRECEDENCE);

                  entry SEND_LOCAL_CONTROL_MESSAGE
                              (TYPE_CONTROL_MESSAGE:in CONTROL_MESSAGES;
                                   LAN_ADDRESS:in LAN_ADDRESSEES;
                                USER_MESSAGE:in LAN_CONTROL_TALK := NONE);

        end SEND_LOCAL_MESSAGE;

        task body SEND_LOCAL_MESSAGE is separate;

End MESSAGE_SERVER;

package body MESSAGE_SERVER is separate;
```

```
package MESSAGE_FORMATS is
        -- This package will be used to encapsulate the data structures
        -- of all known message formats. It will include LAN Data, Control,
        -- Piggybacked and Acknowledgement messages. It may also provide
        -- for canned messages with fields that are pre-initialized.
        -- Such messages might be used, for example, by NC in sending
        -- an Acknowledgement to the source NC. Thi package will be
        -- used by all other packages/modules that handle messages.
        -- While no details of the implementation are provided , most
        -- message formats could use records for ease of reference to
        -- the various fields of a message.

end MESSAGE_FORMATS;

package body MESSAGE_FORMATS is separate;

--------------------------------------------------------------------------------

with    TRANPORT_CONTROL_INPUT_OUTPUT,
        MESSAGE_SERVER,
        BUFFER_MANAGER,
        MESSAGE_FORMATS;


package NATIONAL_COMMUNICATIONS_INPUT_OUTPUT is
        -- This package defines the National Communications Module. This
        -- module serves as the interface between SPIICE modules and LAN
        -- protocols, and the resources and protocols of the DDN. The
        -- module performs sequencing of outgoing or acknowledgement
        -- of incoming messages/message fragments that are part of
        -- Interactive or Deferred sessions. The package is comprised
        -- of control structures to provide a high level of multi-tasking
        -- and incorporates two basic requirements:
        --          (1) Serve Interactive before Deferred Requirements
        --             of the same precedence
        --          (2) Serve Requirements of higher precedence before
        --             those of lower precedence

        use TRANPORT_CONTROL_INPUT_OUTPUT,
            MESSAGE_SERVER,
            BUFFER_MANAGER,
            MESSAGE_FORMATS;


        task NATIONAL_COMMUNICATIONS is
                -- This task is the primary control task for the National
                -- Communications Module. It serves to sequentially
                -- receive messages from TCP(the DDN) or from Message
```

```
                -- Server(the LAN), determine what is the session state
                -- associated with the message (OPEN, ESTABLISHED, CLOSE
                -- or ABORT) and place the message on the proper queue
                -- for processing by the appropriate server task(s).

                entry TCP_RECEIVE_RETURN
                        (LOCAL_CONNECTION:in LOCAL_CONNECTION_NAMES;
                         BUFFER_POINTER:in BUFFER_ADDRESS;
                         RESPONSE_STRING:in TCP_USER_MESSAGE;
                         BYTE_COUNT:in LAN_DATA_MESSAGE.LENGTH;
                         PUSH_FLAG:in TCP_HEADER_MESSAGE.PSH := NOT_PUSHED;
                         URGENT_FLAG:in TCP_HEADER_MESSAGE.URG := NOT_URGENT);

                entry TCP_OPEN_RETURN
                        (LOCAL_CONNECTION:in LOCAL_CONNECTION_NAMES;
                         RESPONSE_STRING:in TCP_USER_MESSAGE);

                entry TCP_SEND_RETURN
                        (LOCAL_CONNECTION:in LOCAL_CONNECTION_NAMES;
                         BUFFER_POINTER:in BUFFER_ADDRESS;
                         RESPONSE_STRING:in TCP_USER_MESSAGE);

                entry TCP_CLOSE_RETURN
                        (LOCAL_CONNECTION:in LOCAL_CONNECTION_NAMES;
                         RESPONSE_STRING:in TCP_USER_MESSAGE);

                entry TCP_ABORT_RETURN
                        (LOCAL_CONNECTION:in LOCAL_CONNECTION_NAMES;
                         RESPONSE_STRING:in TCP_USER_MESSAGE);

                entry LAN_NC_SESSION_MESSAGE
                        (PRECEDENCE:in LAN_DATA_MESSAGE.SERVICE_LEVEL;
                         SESSION_NUM:in NC_SESSION_TABLE.SESSION_NUMBERS;
                         BUFFER_POINTER:in BUFFER_ADDRESS;
                         BYTE_COUNT:in LAN_DATA_MESSAGE.LENGTH);

                entry NC_CONTROL_MESSAGE
                        (TYPE_CONTROL_MESSAGE:in CONTROL_MESSAGES;
                         USER_MESSAGE:in LAN_CONTROL_TALK;
                         BYTE_COUNT:in LAN_DATA_MESSAGE.LENGTH);

        end NATIONAL_COMMUNICATIONS;

        task body NATIONAL_COMMUNICATIONS is separate;

--
```

```
task type OPEN_SERVER_QUEUE is
        -- This task type serves as a type/template for the creation
        -- of tasks that need to use the TCP_OPEN_SERVER task routines.
        -- NC utilizes these tasks to communicate asynchronously with
        -- task TCP_OPEN_SERVER to begin the action of opening a local
        -- connection for a session requirement. The task type is used
        -- to build two different sets of precedece queues, one for
        -- interactive and another for Deferred traffic.

        entry ACCESS_TCP_PORT
                (PRECEDENCE,SESSION_TYPE := INTERACTIVE)
                (LOCAL_PORT:in TRANSMISSION_CONTROL_BLOCK.
                                        LOCAL_SOCKETS;
                FOREIGN_PORT:in TRANSMISSION_CONTROL_BLOCK.
                                        FOREIGN_SOCKETS;
                TYPE_OPEN:in ACTIVE_PASSIVE_FLAG := ACTIVE;
                TIME_OUT:in TIME_TO_LIVE := DEFAULT_LIFE;
                CLASSIFICATION:in TRANSMISSION_CONTROL_BLOCK.
                                        SECURITY := UNCLASSIFIED;
                PRECEDENCE:in TRANSMISSION_CONTROL_BLOCK.
                                        SERVICE_LEVEL := ROUTINE;
                OPTIONS:in TCP_HEADER_MESSAGE.CHOICES := NO_OPTIONS);

end OPEN_SERVER_QUEUE;

task body OPEN_SERVER_QUEUE is separate;

--

task TCP_OPEN_SERVER is
        -- This task performs as an interface module or gateway
        -- to TCP tasks needed to OPEN local connections to some
        -- destination. This task services the OPEN_SERVER_QUEUE
        -- and sequentially processes requests to open connections
        -- and enforces the requirement to service Interactive
        -- sessions before servicing Deferred session requests
        -- (cf equal precedence). Much like the TCP_SERVER_TASK,
        -- this task must obtain and return access to TCP by going
        -- through the PRECEDENCE_QUEUE_SERVER.


        entry OPEN_CONTROL_MESSAGE
                (TYPE_CONTROL_MESSAGE:in CONTROL_MESSAGES);

        entry OPEN_SESSION_MESSAGE
                (PRECEDENCE,SESSION_TYPE := INTERACTIVE)
```

```
                        (LOCAL_PORT:in TRANSMISSION_CONTRCL_BLOCK.
                                                        LOCAL_SOCKETS;
                         FOREIGN_PORT:in TRANSMISSION_CCNTROL_BLOCK.
                                                       FOREIGN_SOCKETS;
                         TYPE_OPEN:in ACTIVE_PASSIVE_FLAG := ACTIVE;
                         TIME_OUT:in TIME_TO_LIVE := DEFAULT_LIFE;
                         CLASSIFICATION:in TRANSMISSION_CCNTROL_BLOCK.
                                              SECURITY := UNCLASSIFIED;
                         PRECEDENCE:in TRANSMISSION_CONTROL_BLOCK.
                                              SERVICE_LEVEL := ROUTINE;
                         OPTIONS:in TCP_HEADER_MESSAGE.CHOICES := NO_OPTIONS);

                entry TCP_OPEN_ACKNOWLEDGE
                        (LOCAL_CONNECTION:in LOCAL_CONNECTION_NAMES;
                         SESSION_NUM:in NC_SESSION_TABLE.SESSION_NUMBERS);

                entry BUFFER_SERVICE_RETURN
                        (BUFFER_POINTER:in BUFFER_ADDRESS);


        end TCP_OPEN_SERVER;

        task body TCP_OPEN_SERVER is separate;

--
        task type TCP_SERVER_QUEUE is
                -- TCP_SERVER_QUEUE serves as a type/template for
                -- the creation of tasks that need to use the TCP_SERVER
                -- task routines. These tasks are utilized by NATIONAL_
                -- CCMMUNICATIONS to communicate asynchronously with
                -- task TCP_SERVER once a session is established. Each
                -- created task represents a message/message fragment.


                entry SESSION_MESSAGE_QUEUE
                        (PRECEDENCE,SESSION_TYPE := INTERACTIVE)
                        (LOCAL_CONNECTION:in LOCAL_CONNECTION_NAMES;
                         SESSION_NUM:in NC_SESSION_TABLE.SESSION_NUMBERS;
                         BUFFER_POINTER:in BUFFER_ADDRESS;
                         BYTE_COUNT:in LAN_DATA_MESSAGE.LENGTH);

                entry CONTROL_SESSION_MESSAGE_QUEUE
                        (TYPE_CONTROL_MESSAGE:in CONTROL_MESSAGES);

                entry RECEIVE_ACKNOWLEDGE_QUEUE
                        (MESSAGE_NUM:in LAN_MESSAGE.MESSAGE_NUMBER;
                        (FRAGMENT_NUM:in LAN_MESSAGE.FRAGMENT_NUMBER);

                entry SEND_ACKNOWLEDGE_QUEUE
```

```
                        (MESSAGE_NUM:in LAN_MESSAGE.MESSAGE_NUMBER;
                        (FRAGMENT_NUM:in LAN_MESSAGE.FRAGMENT_NUMBER);

        end TCP_SERVER_QUEUE;

        task body TCP_SERVER_QUEUE is separate;

--

        task type TCP_SERVER_TASK is
                -- TCP_SERVER_TASK serves as a type/template for
                -- the creation of tasks that need to use the TCP_INPUT_
                -- OUTPUT task routines. Each TCP_SERVER_TASK is associated
                -- with a Local Connection and Session Number, and provides
                -- for a sequential access path to TCP tasks. This task
                -- will send a message fragment/message to TCP and await
                -- an acknowledgement from the destination NC module.
                -- Similarly, this task also sends acknowledgements to
                -- the source NC module for messages/message fragments
                -- received.

                entry SESSION_MESSAGE
                        (PRECEDENCE,SESSION_TYPE := INTERACTIVE)
                        (LOCAL_CONNECTION:in LOCAL_CONNECTION_NAMES;
                         SESSION_NUM:in NC_SESSION_TABLE.SESSION_NUMBERS;
                         BUFFER_POINTER:in BUFFER_ADDRESS;
                         BYTE_COUNT:in LAN_DATA_MESSAGE.LENGTH);

                entry CONTROL_SESSION_MESSAGE
                        (TYPE_CONTROL_MESSAGE:in CONTROL_MESSAGES);

                entry RECEIVE_ACKNOWLEDGE
                        (MESSAGE_NUM:in LAN_MESSAGE.MESSAGE_NUMBER;
                        (FRAGMENT_NUM:in LAN_MESSAGE.FRAGMENT_NUMBER);

                entry SEND_ACKNOWLEDGE
                        (MESSAGE_NUM:in LAN_MESSAGE.MESSAGE_NUMBER;
                        (FRAGMENT_NUM:in LAN_MESSAGE.FRAGMENT_NUMBER);

                entry BUFFER_SERVICE_RETURN
                        (BUFFER_POINTER:in BUFFER_ADDRESS);

        end TCP_SERVER_QUEUE;

        task body TCP_SERVER_QUEUE is separate;
```

```
--
        task PRECEDENCE_QUEUE_SERVER is
                -- PRECEDENCE_QUEUE_SERVER controls access to all
                -- TCP user operations/entries. This is done by
                -- maintaining two sets of service queues (Interactive
                -- and Deferred) for flash, immmediate,
                -- priority, routine, and control message precedences.
                -- In its role as a gatekeeper to TCP, it serves
                -- tasks TCP_OPEN_SERVER, and TCP_SERVER_TASK in trying
                -- to open connections, use established
                -- connections, or close/abort existing connections.
                -- Rendezvous is needed to get access to TCP, and
                -- to release it back to the PRECEDENCE_QUEUE_SERVER.

                entry PRECEDENCE_QUEUE
                        (SERVICE_LEVEL,SESSION_TYPE := INTERACTIVE)
                        (RE_ENTRY_NAME:in TCP_SERVER_TASK_NAMES);

                entry TCP_ACCESS_DONE;

        end PRECEDENCE_QUEUE_SERVER;

        task body PRECEDENCE_QUEUE_SERVER is separate;


--
        task ERROR_HANDLER is
                -- This task will handle TCP user error messages.
                -- It utilizes the other tasks of NC to either
                -- recover from non-fatal errors, or to ABORT
                -- when it detects a fatal error.

                entry NC_ERROR_IN
                        (LOCAL_CONNECTION:in LOCAL_CONNECTION_NAMES;
                         RESPONSE_STRING:in TCP_USER_MESSAGE);

                entry NC_ERROR_OUT
                        (LOCAL_CONNECTION:in LOCAL_CONNECTION_NAMES;
                         RESPONSE_STRING:in TCP_USER_MESSAGE);

        end ERROR_HANDLER;
```

```
        task body ERROR_HANDLER is separate;
--
end NATICNAL_COMMUNICATIONS_INPUT_OUTPUT;
package body NATIONAL_COMMUNICATIONS_INPUT_OUTPUT is separate;
```

APPENDIX D
SPLICE MESSAGE FORMATS

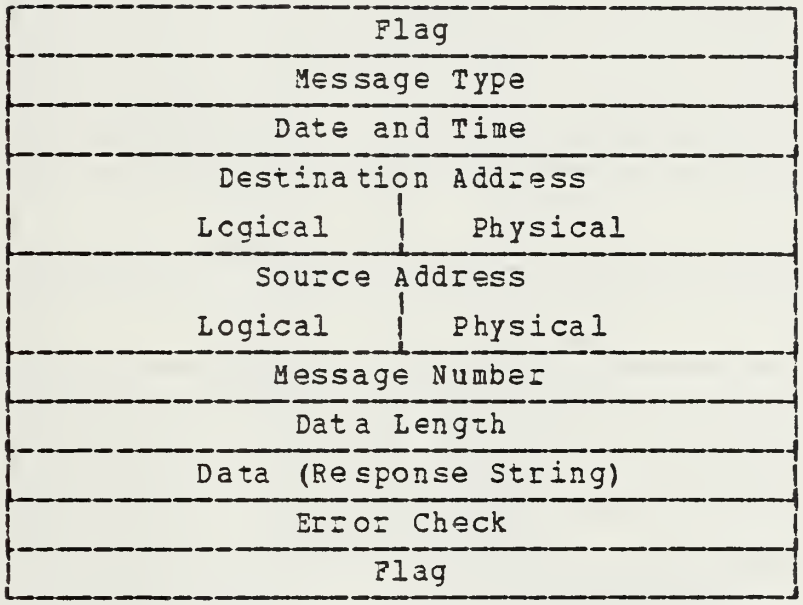


Figure D.1 LAN Control Message.

Note that the data portion of the control message is present to accomodate a response string. This is especially useful when the control message is the result of an error.

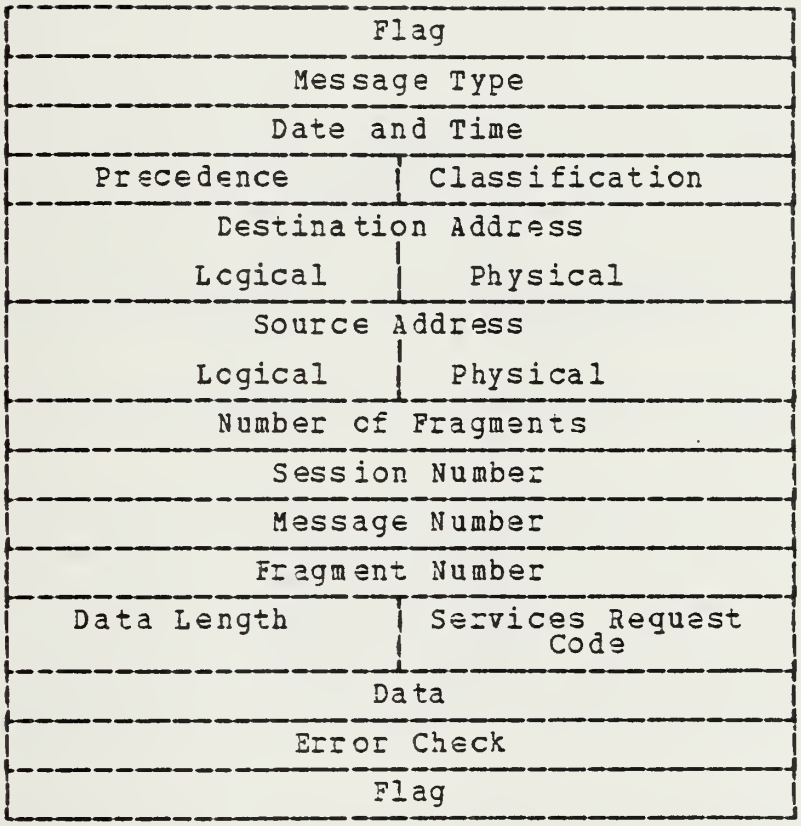


Figure D.2 LAN Data Message.

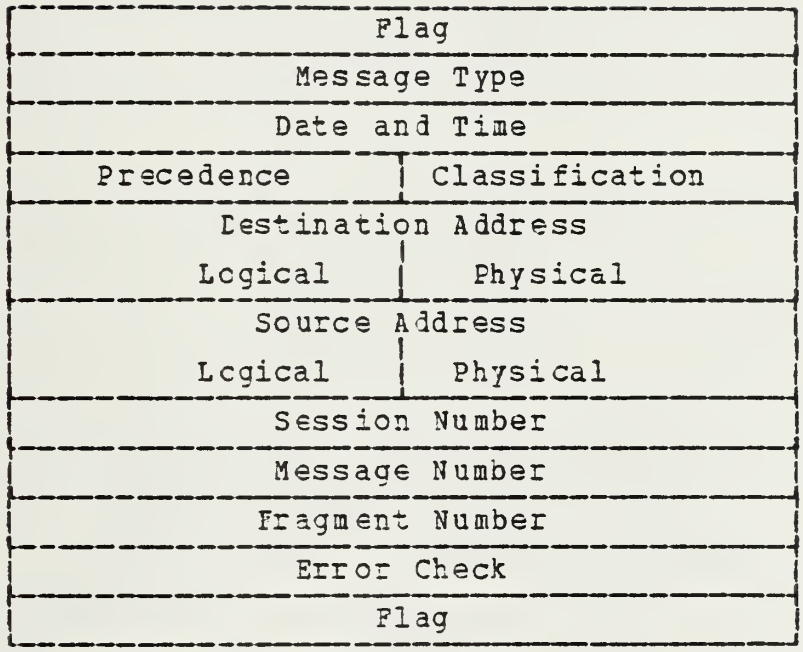


Figure D.3 LAN Acknowledgement Message.

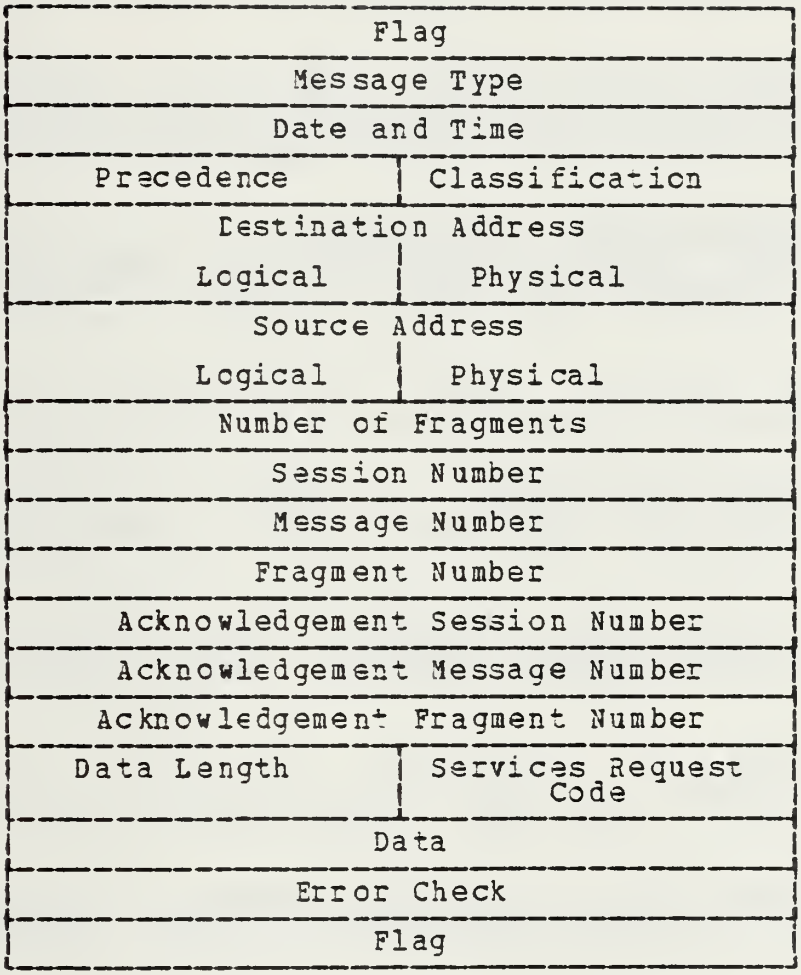


Figure D.4 LAN Piggybacked Data Message.

LIST OF REFERENCES

1. Schneidewind, Norman F., Functional Design of a Local Area Network for the Stock Point Logistical Integrated Communications Environment, Final Report, December 1982.

2. Department of the Navy Fleet Material Support Office, Functional Description F94L0-001-9260-FD-SU01, Stock Point Logistical Integrated Communications Environment (SPLICE), 2 February 1981.

3. Barrett, K.M., Integration Considerations for the Stock Point Logistics Integrated Communications Environment (SPLICE) Local Area Network, Master's Thesis, Naval Postgraduate School, Monterey, California, December, 1982.

4. Defense Communications Agency, Defense Data Network Program Plan, January 1982, Revised May 1982.

5. Schneidewind, Norman F., Methods for Interconnecting Local Area Networks to Long Distance Networks, February 1983.

6. Information Sciences Institute, University of Southern California, DOD Standard Transmission Protocol, RFC-793, Defense Advanced Research Projects Agency, September 1981.

7. Clark, David D., "Modularity and Efficiency in Protocol Implementation," RFC-817, Defense Advanced Research Projects Agency, July 1982.

8. Opel, C.E., Network Management of the SPLICE Computer Network, Master's Thesis, Naval Postgraduate School, Monterey, California, December, 1982.

9. Katzan, Harry Jr., Systems Design and Documentation, Van Nostrand Reinhold Company, New York, New York, 1976.

10. Inman, K.A., Jr., and Marthouse, R.C., Supply Point Logistics Integrated Communications Environment (SPLICE) Local Area Computer Network Design Issues for Communications, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1982.

11. Reinhart, J.N. III and Arana, R., Database and Terminal Management Functional Design Specifications in Support of Stock Point Logistics Integrated Communications Environment (SPLICE), Master's Thesis, Naval Postgraduate School, Monterey, California, June 1982.

12. Parnas, D. L., "On the Criteria to be used for Decomposing Systems into Modules," Communications of the ACM, Vol. 15, #12, December 1972, pp. 1053-1058.

13. Parnas, D. I., "Designing Software for Ease of Extension and Contraction," IEEE Transactions on Software Engineering, Vol. SE-5, #2, March 1979, pp. 128-137.

14. Strazisar, V. "How to Build a Gateway," IEN:109, BBN, August 1979.

15. Information Sciences Institute, University of Southern California, Internet Protocol, RFC-791, Defense Advanced Research Projects Agency, September 1981.

16. Defense Communications Agency Report DCA100-80-C-0037, Evaluation of Ada as a Communications Programming Language, Brintzenhoff, A. L., et al, 31 March 1981.

17. Booch, Grady, Software Engineering with Ada, The Benjamin/Cummings Publishing Company, 1983.

BIBLIOGRAPHY

Department of the Navy Fleet Material Support Office, System System Specification F94L0-001-9260-SS-S001, Stock Point Logistics Integrated Communications Environment (SPLICE), 2 February 1981.

Department of the Navy Solicitation Document N66032-82-R-0007, Acquisition of Hardware, Software and Services to Support the Stockpoint Logistics Integrated Communications Environment (SPLICE) Project at 62 Navy Stock Point Sites, 1 March 1982.

Heiden, Hiedi B., and Duffield, Howard C., Defense Data Network, Defense Communications Agency internal working paper, 1982.

Sheltzer, Alan, Hinden, Robert, and Bresica, MIke, "Connecting Different types of Networks with Gateways", Data Communications, August 1982.

Tanenbaum, Andrew S., Computer Networks, Prentice-Hall, Inc., 1981.

Tanenbaum, Andrew S., "Network Protocols," Computing Surveys, V.13,N.4,pp.453-489, December 1981.

Walker, Steven T., "Department of Defense Data Network", Signal, pp.42-47, October 1982.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Professor Norman F. Schneidewind Code 54SS Department of Administrative Sciences Naval Postgraduate School Monterey, California 93940	2
4.	Commander, Naval Supply Systems Command ATTN: LCDR Dana Fuller, Code 0415A Washington, D.C. 20379	1
5.	Fleet Material Support Office ATTN: LCDR Ted Case Code 94L Mechanicsburg, Pennsyvania 17055	1
6.	Ms. Mary Willoughby P.O. Box 94 Mendocino, California 95460	1
7.	Naval Postgraduate School Computer Technologies Curricular Office Code 37 Monterey, California 93940	1
8.	Department Chairman Department of Computer Science Code 52 Naval Postgraduate School Monterey, California 93940	1
9.	Captain David D. Carlsen HQ, U.S. Army Computer Systems Command Stop H-2 Fort Belvoir, Virginia 22060	2
10.	Captain Dan P. Krebill Computer Systems Division DAAS, Thayer Hall West Point, New York 10996	2
11.	Mr. Paul Cohen Defense Communications Agency Defense Communications Engineering Center 1860 Wiehle Avenue Reston, Virginia 22090	1
12.	Mr. Tony Brintzenhoff Systems Consultants, Inc. 4015 Hancock Street San Diego, California 92110	1

13. Mr. Bill Whitaker 1
WIS JPMO
Washington, D.C. 20330

14. Mr Don Akers 1
NAVCOMMUNITY Washington
ATTN: Code 431
Washington, D.C. 20390

15. Dr. Charles Arnold 2
Naval Underwater Systems Center
New London, CT. 06320

16. CAPT Ted Rogers 1
Box 327
Lumberport, W.V. 26386

17. CPT Mark R. Kindl 1
413 E. Washington St.
Villa Park, Illinois 60181

201630

Thesis
C228 Carlsen
c.1 The national communications module of the stock point logistics integrated communications environment (SPLICE) local area networks.

8 MAR 87 31760

201630

Thesis
C228 Carlsen
c.1 The national communications module of the stock point logistics integrated communications environment (SPLICE) local area networks.

thesC228
The national communications module of th
3 2768 000 67627 4
DUDLEY KNOX LIBRARY